ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ПОЛІТЕХНІЧНИЙ УНІВЕРСИТЕТ
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Кафедра комп’ютерних інтелектуальних систем та мереж

ШЕВЧУК Данило Олегович

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ПРОГРАМНА СИСТЕМА ТРАСУВАННЯ МАРШРУТІВ НА ОСНОВІ МЕТОДІВ ФРОНТУ ХВИЛІ

Спеціальність 123 – Комп’ютерна інженерія
Спеціалізація - Комп’ютерні системи та мережі

Керівник: Защолкін Костянтин Вячеславович,
кандидат технічних наук, доцент

Одеса – 2020

АНОТАЦІЯ

Шевчук Д. О. Програмна система трасування маршрутів на основі методів фронту хвилі – кваліфікаційна робота бакалавра. Одеса, 2020: N с., N рис., N табл., 1 додаток, N джерел.

У дипломній роботі розглянуті і проаналізовані алгоритми пошуку маршрутів, обґрунтовані переваги хвильового методу та виконана реалізація ПЗ згідно алгоритму. Розроблено методику ініціалізації, поширення хвилі і відновлення шляху. Програма для методики знаходження найкоротшого шляху розроблена на мові програмування JavaScript.

Програмна система реалізує знайдення маршрутвів з фрагментами більшого чи меншого приорітету. За допомогою даного програмного продукту можна видати найкоротший шлях з великою ефективністю за рахунок принципу зворотного трасування.

ТРАСУВАННЯ, ТОЧКА СТАРТУ, ТОЧКА ФІНІШУ, НАПРЯМОК ОБХОДУ, ПЕРЕШКОДИ, ЗОНИ ПРИОРІТЕТУ, JAVASCRIPT.

SUMMARY

Shevchuk D. O. Routing software system based on wave front methods – qualifying work of the bachelor. Odesa, 2020: N pages, N images, N tables, 1 addition, N source links.

The purpose of the thesis is to develop software system for finding the shortest path when tracing routes using a wave front algorithm.

The thesis considers and analyzes the algorithms of search with Depth-first search and method of the wave front. A method of initialization, wave propagation and path recovery has been developed. The program for the method of finding the shortest path is developed in the JavaScript programming language.

The software system implements finding routes with fragments with higher or lower priority. With this software product, you can give the shortest path with high efficiency due to the principle of reverse tracing.

TRACE, START POINT, TARGET POINT, BYPASS DIRECTION, OBSTACLES, PRIORITY ZONES, JAVASCRIPT.

ЗМІСТ

ВСТУП

Пошук шляху — це знаходження варіанту з найкоротшим шляхом між двома точками за допомогою комп'ютерної програми. Задача пошуку шляху пов'язана з задачею про найкоротший шлях у рамках теорії графів, яка розглядає знаходження шляху, що найкраще відповідає таким критеріям як: найкоротший, найдешевший, найшвидший і так далі, між двома точками у великій системі.

Хвильовий алгоритм – категорія алгоритмів, які застосовуються у випадках де потрібно знайти мінімальний шлях у аланарному графі, тобто двомірній матриці, із однаковою довжиною ребер. Ця категорія алгоритмів була заснована завдяки алгоритму пошуку в ширину.

Хвильовий алгоритм оперує елементи графу подібно алгоритму пошуку в ширину, але при цьому допустимо що елементи графу можуть бути перешкодами, тобто шлях не може проходити через цей елемент. У цьому випадку шлях повинен поширюватися в усіх напрямках, окрім тих, де наступний елемент є перешкодою.

Хвильові алгоритми прийнято поділяти за двома типами поширювання: ортоганальний та діагонально-ортогональний. Кожен крок у першому із цих двох повинен поширювати хвилю тільки у елементи які мають пову сусідню межу, тобто різниця координат цих елементів дорівнює одиницею тільку у одній осі координат, наприклад сусідній елемент до елементу із координатами (1, 1) (x та y відповідно) у цьому випадку є (0, 1), (1, 0), (2,1) та (1, 2).

1 АНАЛІТИЧНИЙ ОГЛЯД ЗА ТЕМОЮ ДИПЛОМНОЇ РОБОТИ

У даному розділі розглянуто математичний опис завдання пошуку маршрутів.

1.1 Постановка задачі трасування маршрутів

Назва алгоритму «хвильовий» – пов’язана з тим, що визначення індексів n вершин графу H відбувається, як шлях з початкової вершини v певної хвилі, яка спрямовується за напрямком дуг. Алгоритм закінчить свою роботу коли хвиля дійде до кінцевої вершини u. Значення індексу, «принесеного хвилею», у вершині u буде дорівнювати довжині знайденого маршруту. А для визначення маршруту потрібно з кінцевої вершини u повертатися в зворотному напрямку і помічати послідовно одну довільну вершину зі значеннями індексу n–1, n–2, …, 1, 0. Зрозуміло, що вершина з індексом 0 – це початкова вершина v. Вершини u1, u2, …, un–1, взагалі, можуть бути визначені неоднозначно. Неоднозначність відповідає випадкам, коли існує декілька різних мінімальних шляхів з v до u в орграфі G. Для процесу тестування впроваджуються різні методи.

Наприклад, визначаємо мінімальний шлях з v1 до v6 в орієнтованому графі G, заданому матрицею суміжності (відповідний граф представлено на рис. 1.1)

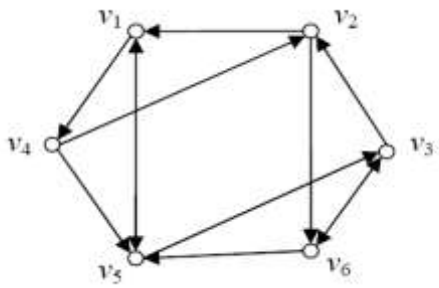


Рисунок 1.1 – Приклад роботи хвильового алгоритму

Діючи згідно з алгоритму, послідовно знаходимо F1(v1) = {v4, v5}; F2(v1) = D(F1(v1)) \ {v1, v4, v5} = {v2, v3}; F3(v1) = D(F2(v1)) \ {v1, v2, v3, v4, v5} = {v6}. Таким чином, v6 ∈ F3(v1), а, отже, за пунктом 3 існує шлях з v1 до v6 завдовжки 3 - це і є мінімальним шляхом. Тепер знайдемо мінімальний шлях із v1 до v6. Визначимо множину F2(v1) ∩ D -1(v6) = {v2, v3} ∩ {v2, v3} = {v2, v3}. Виберемо будь-яку вершину зі знайденої множини, наприклад, v3. Визначимо далі множину F1(v1) ∩ D -1(v3) = {v4, v5} ∩ {v4, v5, v6} = {v4, v5}. Далі виберемо будь-яку вершину зі знайденої множини, наприклад, v5. Тоді (v1, v5, v3, v6) –мінімальний шлях з v1 до v6 в орієнтованому графі G, а відстань між v1 та v6 дорівнює 3. Очевидно, хвильовий алгоритм може застосовуватися не тільки для орієнтованих, а й для неорієнтованих графів. В останньому випадку, пересування з однієї вершини до іншої можливі в обидві сторони. Хвильовий алгоритм широко застосовується і у розробці комп'ютерних ігор, наприклад, коли необхідно визначити оптимальний маршрут пересування гравця або певного „юніта" з однієї точки карти до іншої. Наведемо приклад такої задачі: нехай потрібно знайти найкоротший шлях з точки A до точки B на карті, яка зображена на рис. 3, а. На ній заштриховані елементи відповідають певним перешкодам на шляху, тобто в цих частинах місцевості гравець не може пройти. Також будемо вважати, що пересування гравця може бути реалізовано тільки по вертикалі та горизонталі. Відповідний цієї карті граф представлено на рис. 1.2, б.

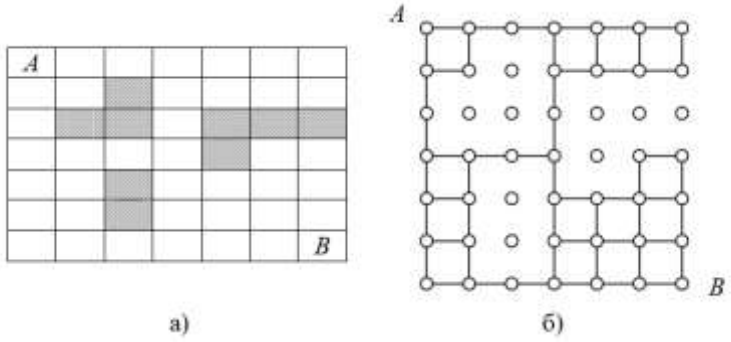


Рисунок 1.2 – Приклад карти території та відповідного графу

Після циклу хвильового алгоритму отримаємо наступні індекси вершин – елементів карти (рис. 1.3, а). На рис. 1.3, б зображено два із найдених

маршрутів з вершини A у вершину B. Можна помітити, довжина знайденого маршруту дорівнює 12.

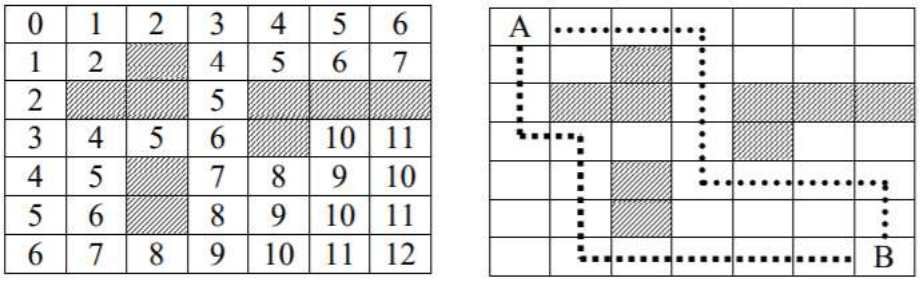


Рисунок 1.3 – Робота хвильового алгоритму для графу

Можна оновити процес пошуку найкоротшого маршруту за допомогою хвильового алгоритму та зробити його більш економічним та швидким. Для цього потрібно розповсюджувати хвилю з початкової вершини A (перша хвиля), та кінцевої B (друга хвиля). Щоб відрізняти індекси хвиль - індекси останньої позначимо зі штрихом. Алгоритм вважають закінченним, коли обидві хвилі зустрічаються (рис. 1.4).

0	1	2	3	4	5	6
1	2		4	5	6	
2			5			
3	4	5	6/6’		4’	3’
4	5		5’	4’	3’	2’
5	6/6’		4’	3’	2’	1’
6/6’	5’	4’	3’	2’	1’	0’

Рисунок 1.4 – Двонаправлений хвильовий алгоритм

Елементи матриці, де дві хвилі зустрічаються, позначені подвійними лініями. З порівняння видно, що знайдені найкоротші маршрути співпадають. Хоча в цьому випадку економія складає всього один елемент, але можна зрозуміти, що на більш складних картах робота модифікованого хвильового алгоритму буде ефективнішою за простий хвильовий алгоритм.

1.2 Область використання трасування маршрутів

У даному підрозділі розглянуті області використання трасування маршрутів.

1.2.1 Використання трасування маршрутів у системах проектування печатних платформ

Трасування друкованих плат — це один з етапів проектування радіоелектронної апаратури і полягає він в покроковому проектуванні структури провідників друкованої плати вручну або з використанням однієї з САПР друкованих плат.

Трасування з'єднань, як правило, завершальний етап конструкторського проектування РЕА, який полягає у визначенні топології та геометрії провідників, що з'єднують контакти елементів проектованого пристрою.

Задача трасування — є однією з найбільш трудомістких в загальній проблемі автоматизації проектування РЕА. Це пов'язано з різноманітними факторами способів конструктивно-технологічної реалізації з'єднань, для кожного з яких, при алгоритмічному вирішенні задачі застосовуються специфічні критерії оптимізації та обмежень. З математичної точки зору, задача трасування — оптимізаційна задача, яка полягає у виборі раціонального рішення з величезної кількості можливих варіантів.

Кількість можливих варіантів в задачах трасування така, що для відносно нескладних систем вирішення шляхом перебору всіх можливих варіантів з'єднань - неможлива. Саме тому представляють інтерес методи трасування, які шукають не найкращий варіант трасування (що потребує гарантованого перебору та оцінки всіх варіантів), а раціональний варіант, який незначно поступається оптимальному, але буде знайдений за відносно невеликий час.

Основна задача трасування формується в такий спосіб: за заданою схемою з'єднань прокласти необхідні провідники на площині (платі, кристалі і т.п.) таким чином, щоб досягти зазначеного критерію якості трасування з

урахуванням заздалегідь заданих обмежень. Основними є обмеження на ширину провідників і мінімальні відстані між ними.

Хвильові алгоритми, засновані на ідеях Лі та розроблені Ю. Л. Зіманом і Г. Г. Рябовим. Дані алгоритми набули широкого поширення в існуючих САПР, оскільки вони дозволяють легко враховувати технологічну специфіку друкованого монтажу зі своєю сукупністю конструктивних обмежень. Ці алгоритми завжди гарантують побудова траси, якщо шлях для неї існує.

1.2.2 Використання трасування маршрутів у комп'ютерних іграх

Пошук шляху в комп'ютерних іграх це одна з проблем ігрового штучного інтелекту. Для її вирішення досліджуються та використовуються різні алгоритми пошуку найкоротшого шляху.

Невід'ємною частиною сучасних комп'ютерних ігор є наявність ігрового штучного інтелекту (ШІ). Хоча підхід до ігрового ШІ серйозно відрізняється від підходу до традиційного ШІ без нього обійтися вже важко. Одним з яскравих прикладів ігрового штучного інтелекту є створення ігрового персонажу – програми-робота, що імітує партнерів в грі, в мережевих поєдинках, командних боях і т. п. Програма створення ігрового персонажу заснована на модулі штучного інтелекту, який адаптований до особливостей гри: мапі, правил, а також до типу гри. Незалежно від типу штучного персонажу та його призначення первинною задачею є його переміщення. Для руху по відкритій території, наприклад, з точки А в точку Б можна дістатися по прямій лінії. Але якщо рух відбувається в приміщенні з кімнатами, або виникають перешкоди, то потрібно вже більш складний підхід. У найпростішому випадку приміщення має прямокутну форму і розбито на квадратні клітини (комірки) одного розміру, сторони яких паралельні координатним осям. Всі клітини поділяються на прохідні – білі і непрохідні (стіни, перешкоди) – чорні. З кожної клітини є можливість переміститься тільки в сусідні клітини лише по горизонталі або по вертикалі. Зазвичай

приміщення складається з залів (кімнат), які з'єднуються коридорами. У залах можуть перебувати перешкоди, які необхідно обходити. Припустимо, що в кожному прохідному залі побудовані найкоротші внутрішні траси, що з'єднують різні входи. Побудовані траси – сегменти геометричного графа: його вершини відповідають точкам входу, а ребра - побудованим трасам. Даний граф, по суті, є дорожньою картою (roadmap). Далі вирішується задача пошуку найкоротших маршрутів по дорожній карті.

До найбільш популярних алгоритмів пошуку маршруту в графі можна віднести: – Пошук в ширину (англ. Breadth-First Search, BFS) дозволяє обчислити найкоротші відстані (в термінах кількості ребер) від виділеної вершини орієнтованого графа до всіх інших вершин, і / або побудувати кореневе спрямоване дерево, відстані в якому збігаються з відстанями в вихідному графі . Крім того, пошук в ширину дозволяє вирішувати задачу перевірки досяжності (чи існують шляхи між вершиною джерелом і іншими вершинами графа).

1.2.3 Використання трасування маршрутів у робототехніці

Одна з основних функцій робота полягає у виконанні руху в задану точку в залежності від геометричних форм перешкод. Однак не завжди трасування ставить перед собою мету знайти найкоротший шлях; найчастіше це просто неможливо. Програмування даної функції впирається в проблему, яка зводиться до алгоритму обходу перешкод . В основу програм трасування можуть бути покладені алгоритми:

- з дискретним поданням зони руху;
- засновані на теорії графів;
- засновані на методах потенційних полів;
- засновані на евристичних моделях;
- з поданням інформації у вигляді рівнянь.

Розглянемо найпростішу задачу трасування: обхід двовимірних лабіринтів по заданій цифровий (растрової) карті місцевості у вигляді двовимірної площині, розділеної на клітини (квадратні, трикутні або гексагональних), з зазначеними точкою старту (точки, з якої ми повинні провести маршрут), точкою цілі (точка, в яку ми повинні провести маршрут) і перешкодами різної геометричної форми. Спочатку необхідно вибрати стратегію обходу перешкоди. Ми можемо піти двома шляхами: перший - прокладати шлях «на ходу», ігноруючи перешкоди до зіткнення з ними; другий - заздалегідь спланувати шлях до початку переміщення. Найбільш прості алгоритми засновані на припущенні, що перешкоди можна «торкатися рукою» і слідувати його контуру, поки вона не буде обійдена. Спочатку вибираємо напрям для руху до мети. рухаємося до зіткнення з перешкодою. При зіткненні вибираємо інший напрямок в Відповідно до стратегії обходу, наприклад: - переміщення у випадковому напрямку. Робимо невеликий зсув в випадковому напрямку при зустрічі перешкоди;

- трасування навколо перешкоди. При зустрічі перешкоди йдемо по його контуру до певного моменту, що визначається евристикою (наприклад кількість переміщень в одному напрямку; зупинити обхід перешкоди при можливості пересування в напрямку, який був бажаним на початку трасування).

При реалізації багатьох алгоритмів на графах виникає необхідність організувати систематичний перебір вершин графа, при якому кожна вершина проглядається точно один раз. Такий перебір можна організувати двома способами: пошуком в глибину або пошуком в ширину.

1.2.4 Використання трасування маршрутів в задачах навігації

Алгоритми знаходження найкоротшого шляху на графі застосовуються для знаходження шляхів між фізичними об'єктами на картографічних сервісах, таких як карти Google або OpenStreetMap.

Оптимізація маршруту по мережі доріг є вирішеним завданням та широко використовується в сучасних навігаційних системах. Проте, це завдання істотно відрізняється від завдання прокладення оптимального маршруту в умовах бездоріжжя, оскільки припускає ряд обмежень. При проектуванні мереж з динамічно змінюваною топологією це завдання набуває особливої актуальності при відсутності навігаційних (векторних) карт. Використання методів варіаційного числення при рішенні поставленої проблеми пов’язано зі значними математичними труднощами, що обумовлює вибір дискретних методів теорії графів і дослідження операцій як альтернативу континуальним методам. Один з найефективніших рішень задачі трасування маршрутів руху мобільних вузлів мережі є пошук оптимального шляху на основі алгоритму методів фронту хвилі.

1.3 Алгоритми пошуку маршрутів

У даному підрозділі детально описано 4 найбільш використовуваних методів трасування маршрутів.

1.3.1 Алгоритм Дейкстри

Алгоритм Дейкстри – полягає у знаходженні найкоротшого шляху від однієї з вершин графа до всіх інших. Алгоритм працює тільки для графів без ребер з негативною вагою;

Розглянемо деякий орієнтований граф, для якого потрібно знайти найкоротший маршрут від вершини 1 до всіх інших вершин. Для розв'язку задач такого типу доцільно використовувати алгоритм Дейкстри.

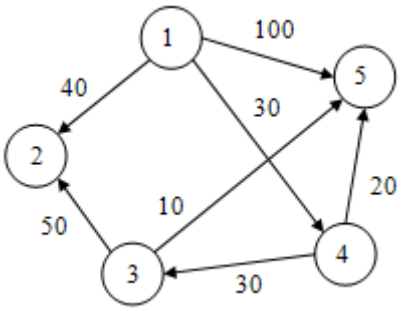


Рис. 1.5 - Приклад роботи алгоритму Дейкстри

Слдуючи алгоритму, початковій вершині (вершина під номером 1) присвоюється постійна мітка Алгоритму Дейкстри після чого переходимо до першого етапу.

Етап Перший: з вершини 1 існує орієнтоване ребро до вершин 2, 4 і 5. Обчислимо для даних вершин відповідні мітки. В результаті отримаємо наступну таблицю.

Номер вершини	Мітка	Статус мітки
1	[0,–]	Постійна
2	[0 + 40,1] = [40,1]	Тимчасова
4	[0 + 30,1] = [30,1]	Тимчасова
5	[0 + 100,1] = [100,1]	Тимчасова

Таблиця. 1.1 - Приклад роботи алгоритму Дейкстри

Серед вершин з тимчасовими мітками, вибиремо ту, значення відстані для якої є найменшим. В нашому випадку такою є вершина 4, з відстанню algoritm_dejkstru_pruklad4. Міняємо статус даної вершини на «постійна» і переходимо до другого етапу.

Етап Другий: з четвертої вершини є орієнтоване ребро до вершин 3 і 5. Обчислюємо для них мітки, після чого таблиця набуде наступного вигляду.

Таблиця. 1.2 – Приклад роботи алгоритму Дейкстри

Номер вершини	Мітка	Статус мітки
1	[0, −]	Постійна
2	[0 + 40,1] = [40,1]	Тимчасова
4	[0 + 30,1] = [30,1]	Постійна
5	[30 + 20,4] = [50,4]	Тимчасова
3	[30 + 30,4] = [60,4]	Тимчасова

В даній таблиці, тимчасову мітку [100, 1] для вершини 5, яку отримали на попередньому етапі, замінено на нову [50, 4], також тимчасову. Це говорить про те, що до вершини 5 знайдено коротший маршрут, який проходить через вершину 4. Після чого, аналогічно першому етапу, вибираємо вершину значення відстані для якої є найменшим (вершина під номером 2) і міняємо її статус на постійну.

Етап Третій: від вершини 2 не можливо потрапити до інших вершин графа, тому на даному етапі ми отримаємо аналогічну другому етапу таблицю, тільки статус вершини 2 змінено на постійна.

Таблиця. 1.3 - Наочний приклад роботи алгоритму Дейкстри

Номер вершини	Мітка	Статус мітки
1	[0, −]	Постійна
2	[0 + 40,1] = [40,1]	Постійна
4	[0 + 30,1] = [30,1]	Постійна
5	[30 + 20,4] = [50,4]	Тимчасова
3	[30 + 30,4] = [60,4]	Тимчасова

Серед вершин, статус мітки яких не дорінює постійна, вибираємо ту, для якої значення відстані Алгоритм Дейкстри є найменшим. Тобто вершину 5, для якої Алгоритм Дейкстри. Міняємо її статус на постійну і переходимо до четвертого етапу.

Етап Четвертий: з вершини 5 не існує орієнтованих ребер до інших вершин графа, тому таблиця міток залишається незмінною, тільки статус вершини 5 замінено на постійну.

Таблиця 1.4 – Приклад роботи алгоритму Дейкстри

Номер вершини	Мітка	Статус мітки
1	[0, −]	Постійна
2	[0 + 40,1] = [40,1]	Постійна
4	[0 + 30,1] = [30,1]	Постійна
5	[30 + 20,4] = [50,4]	Постійна
3	[30 + 30,4] = [60,4]	Тимчасова

На четвертому етапі, ми отримали таблицю, в якій з тимчасовою міткою залишилась тільки вершина 3. І виходячи з того, що з даної вершини можна потрапити у вершину 2 або 5, статус яких становить — постійна, то на цьому процес обчислень за алгоритмом Дейкстри закінчується.

Таким чином ми отримали маршрут найкоротшої довжини, який починається у вершині 1 і обходить всі вершини заданого графа.

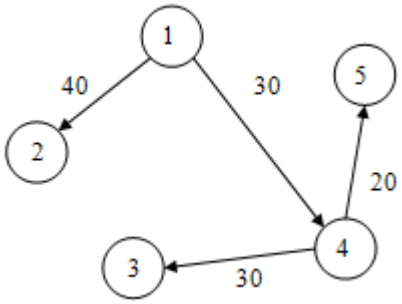


Рисунок 1.6 – Приклад роботи алгоритму Дейкстри

1.3.2 Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда - знаходить найкоротші шляхи від однієї вершини графа до всіх інших у **зваженому** графі. Вага ребер може бути негативною.

Запишемо алгоритм Беллмана-Форда більш детально. Розглянемо деякий орієнтований граф H із зваженими ребрами, без циклів від'ємної довжини. І припустимо, що потрібно знайти найкоротші шляхи від виділеної вершини a до всіх інших вершин даного графа:

Перед початком виконання алгоритму всі вершини графа - непройденими, а ребра - не переглянуті. Кожній вершині в ході виконання алгоритму присвоюємо число d_x , яке рівне довжині найкоротшого шляху з вершини a у вершину x, включаючи тільки пройдені вершини. На першому кроці покладаємо $d_a = 0$ і $d_x = \infty$ для сіх x, відмінних від a. Також, на даному кроці, вершині a присвоюємо мітку «пройдена» і припускаємо $y = a$ (y — остання з пройдених вершин).

Для кожної з вершин графа H наступним чином перерахувати величину $d_x = \min\{d_x, d_y + m_{yx}\}$ (1) : (де m_{yx} - вага ребра (y, x)). Якщо $d_x = \infty$ для всіх непройдених вершин x, процедуру алгоритму Беллмана-Форда необхідно звершити (у вихідному графі відсутні шляхи з вершини у непройдені вершини a). В іншому випадку, мітку «пройдена» необхідно присвоїти тій з вершин x, для якої величина d_x - найменьша. Крім того, ребро, що веде до обраної на даному етапі вершині x вважається переглянутим (для цієї дуги досягався мінімум відповідно до виразу (1)). Після цього, поклавши $y = x$, ітераційний процес продовжується далі. Відмітимо, що якщо для деякої пройденої вершини x відбувається зменшення величини d_x, то з цієї вершини і инцидентного їй переглянутого ребра відповідні мітки знімаються.

Процедура алгоритму Беллмана-Форда завершується тільки тоді, коли всі вершини графа H пройдені і коли після чергового виконання кроку номер два жодне з чисел d_x не змінило свого значення.

1.3.3 Алгоритм пошуку A *

Алгоритм пошуку A * - використовується для знаходження маршруту з найменшою вартістю від однієї вершини (початкової) до іншої (цільової, кінцевої), використовуючи алгоритм пошуку по першому найкращому збігу на графі.

Алгоритм працює з допоміжною функцією (евристикою), аби скеровувати напрям пошуку та скорочувати його тривалість. Алгоритм повний в тому, що завжди знаходить оптимальний розв'язок, якщо він існує.

Алгоритм A* спершу починає відвідувати ті вершини, які ймовірно ведуть до найкоротшого шляху. Для розпізнавання таких вершин, кожній відомій вершині x підставляється значення f(x), яке дорівнює довжині найкоротшого шляху від початкової вершини до кінцевої, який пролягає через обрану вершину. Вершини з найменшим значенням f обирають в першу чергу.

Функція f(x) для вершини x визначається так (1.1):

$$f(x) = g(x) + h(x) \qquad (1.1)$$

де:

– g(x) функція, значення якої дорівнюють вартості шляху від початкової вершини до x,

– h(x) евристична функція, оцінює вартість шляху від вершини x до кінцевої.

Використана евристика не повинна дати завищену оцінку вартості шляху. Прикладом оцінки може служити пряма лінія, а загальний шлях не може бути коротшим за цю лінію.

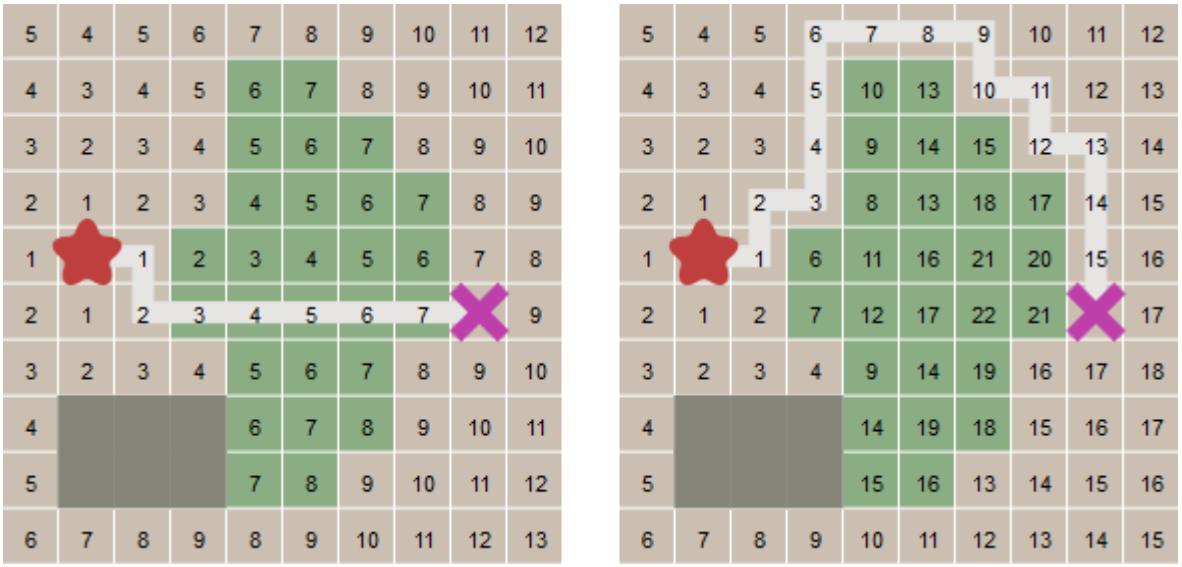


Рисунок 1.7 – Приклад роботи алгоритму пошуку A *

Алгоритм ділить вершини на три класи:

– невідомі вершини: вершини, що ще не були знайдені. Ще невідомий шлях до них. На початку роботи алгоритму всі вершини, окрім початкової, належать до класу невідомих.

– відомі вершини (OpenList): вже відомий (можливо неоптимальний) шлях до цих вершин. Всі відомі вершини та значеннями f зберігаються у списку. З цього списку вибирають найперспективніші вершини. Реалізація цього списку має суттєвий вплив на швидкодію алгоритму, і зазвичай має

вигляд пріоритетної черги (наприклад: бінарна купа). На початку роботи алгоритму - до відомих відносимо тільки початкову вершину.

– повністю досліджені вершини (ClosedList): до цих вершин найкоротший шлях вже відомий. Повністю досліджені вершини додаються до замкнутого списку, аби запобігти багаторазовому дослідженню вже досліджених вершин. Список повністю досліджених вершин на початку роботи алгоритму - порожній.

Кожна відома або повністю досліджена вершина має вказувати на попередні вершини. Завдяки цьому, можна пройти одним шляхом від цієї вершини до початкової.

Коли вершина x буде повністю досліджена, суміжні з нею вершини увійдуть до списку відомих вершин, а сама вершина додається в список повністю досліджених. Вказівники на попередню вершину встановлюють на x. Суміжні вершини, які вже знаходяться в списку повністю досліджених вершин, до списку відомих не додаються, а зворотні вказівники не змінюються. Суміжні вершини, які вже знаходяться в списку відомих - оновлюються (значення f та вказівник на попередню вершину), якщо знайдений до них шлях коротший за відомий.

Коли кінцева вершина потрапляє до списку повністю досліджених вершин - алгоритм зупиняється. Знайдений шлях відтворюється за допомогою вказівників на попередню вершину. Якщо список відомих вершин порожніє - алгоритм припиняє пошук, тому що розв’язку не існує.

Відтворений за зворотними вказівниками знайдений шлях починається з кінцевої вершини та прямує до початкової. Для того, щоб одразу отримати шлях в правильному напрямі, з початкової вершини до кінцевої, в умовах задачі слід переставити місцями початок та кінець. Якщо шукати шлях з кінцевої вершини, відтворений список починатиметься з початкової вершини й прямуватиме до кінцевої.

1.3.4 Алгоритм Флойда-Уоршелла

Алгоритм Уоршелла – базується на порівняні всіх можливих шляхів в графі між кожною парою вершин. Він виконується з $\Theta(|V|^3)$ порівнянь. Це доволі примітивно, враховуючи, що в графі може бути до $\Omega(|V|^2)$ ребер, та кожна комбінація буде перевірена. Він виконує це шляхом поступового поліпшення оцінки по найкоротшому шляху між двома вершинами, до того, поки оцінка не стає оптимальною.

Розгляньмо граф H з ребрами V, пронумерованими від 1 до N. Також розгляньмо функцію shortestPath(i, j, k), яка повертає найкоротший шлях від i до j, використовуючи вершини з множини {1,2,…,k} як внутрішні у шляху. Тепер, нам потрібно знайти найкоротший шлях від кожного i до кожного j, використовуючи тільки вершини від 1 до k + 1.

Для кожної з цих пар вершин, найкоротший шлях може бути або (1) - шлях, у якому є тільки вершини з множини {1, …, k}, або (2)- шлях, який проходить від i до k + 1 а потім від k + 1 до j. Найкоротший шлях від i to j that only uses vertices 1 через k визначається функцією shortestPath(i, j, k), і якщо є коротший шлях від i до (k + 1 до j), то довжина цього шляху дорівнює сумі (конкатенації) найкоротшого шляху від i до k + 1 (використовуючи вершини{1, …, k}) і найкоротший шлях від k + 1 до j (також використовуючи вершини з {1, …, k}).

Ця формула - основна частина алгоритму Флойда-Уоршелла. Алгоритм спочатку обчислює shortestPath(i, j, k) для всіх пар(i, j), де k = 1, потім k = 2, і т. д. Цей процес продовжується, поки k = N, і поки не знайде всі найкоротші шляхи для пар (i, j).

Алгоритм виконується на рисунку 1.8.

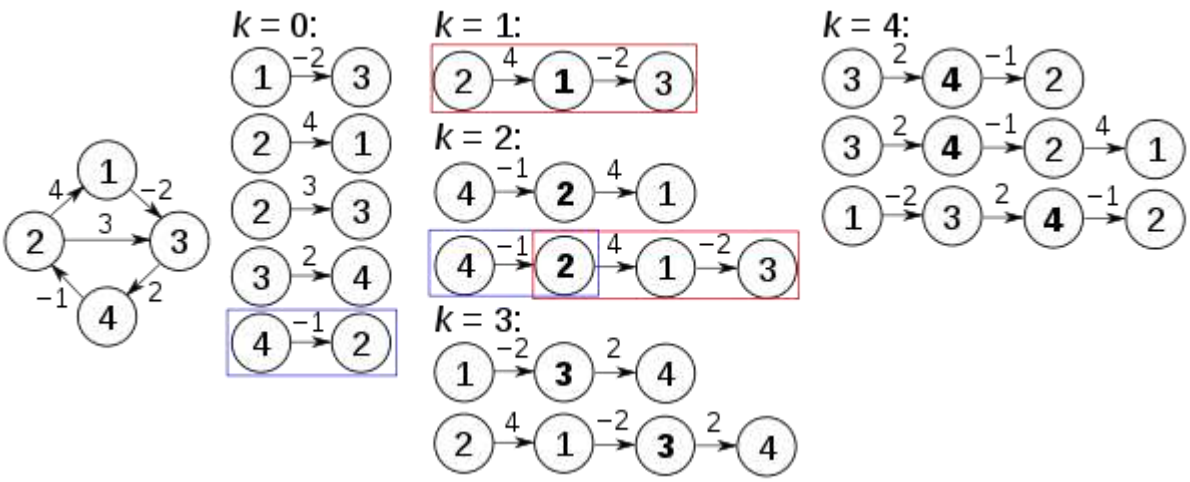


Рисунок 1.8 – Приклад роботи алгоритму Флойда-Уоршелла

Перед першою ітерацією циклу k=0, а відомі шляхи відповідають одиночним ребрам у графі. Коли k=1, знайдені шляхи, які проходять через вершину 1, зокрема: шлях 2→1→3 , замінить шлях 2→3, що проходить через меншу кількість ребер, але є довшим. При k=2, знаходять шляхи, що проходять через вершини {1,2} . Червоні та блакитні квадратики показують, як шлях 4→2→1→3 складається з 4→2 і 2→1→3, визначеними на попередніх ітераціях. Шлях 4→2→3 не розглядається, бо 2→1→3 поки що найкоротший шлях. При k=3,знаходяться шляхи, що проходять через {1,2,3}. Нарешті, при k=4, знайдено всі найкоротші шляхи.

1.4 Висновки

В даному розділі дипломної роботи було описано математичну постановку задачі трасування маршрутів, визначена область використання теми:

- у системах проектування печатних платформ;
- комп'ютерних іграх;
- у робототехніці;
- навігаційних задачах.

Детально описані найпоширеніші алгоритми пошуку найближчої відстані з прикладами та ілюстраціями конкретних задач. Ці алгоритми мають як переваги так і недоліки, але темою моєї дипломної роботи було вибрано саме алгоритм трасування маршрутів на основі методів фронту хвилі. Оскільки, на моє глибоке переконання, саме цей алгоритм – найбільш оптимізований під цільові призначення.

2 ПОСТАНОВКА ТА АНАЛІЗ ЗАВДАННЯ ДИПЛОМНОЇ РОБОТИ

2.1 Завдання на розробку програмної системи трасування маршрутів на основі методів фронту хвилі

2.1.1 Призначення та галузі використання розробки

В дипломному проекті потрібно розробити програмну систему, призначену трасування маршрутів на основі методів фронту хвилі.

Трасування маршрутів хвильовим методом може знаходити застосування в наступних задачах:

– для трасування печатних плат;

– для трасування маршруту робототехніки;

– для пошуку шляху штучного інтелекту у комп’ютерних іграх.

В даному дипломному проекті потрібно реалізувати алгоритм пошуку шляху хвильовим методом.

Основними аналогами програмної системи пошуку шляху хвильовим методом, реалізованої в даному дипломному проекті є по-перше професійні системи трасування печатних плат, по-друге певними аналогами розробки є алгоритми що застосовують при створенні штучного інтелекту комп’ютерних іграх або у ротоботехниці. Головним недоліком професійних систем пошуку шляху хвильовим методом, як і алгоритмів для штучного інтелекту є їх вузька спеціалізація. Інакше кажучи немає можливості знайти шлях для робота-вантажника у системі для трасування печатних плат. Також професійні системи більше підходять для створення комплексних, багатошарових печатних плат. Саме через це використання професійних систем є надто тяжким для користувача без досвіду.

Розроблена програмна система пошуку шляху хвильовим методом може знайти застосування в складі: простих та одношарових печатних плат,

трасування маршруту робототехніки вантажного типу, пошуку шляху у комп’ютерних іграх із двома мірами свободи руху.

2.1.2 Мета та задачі розробки

Мета даного дипломного проекту полягає в створенні програмної системи пошуку шляху хвильовим методом. Завдяки використанню розробленої системи передбачається отримання наступних практичних результатів:

– можливість знаходити шлях на двомірній матриці із перешкодами;

– можливість визначити деякі елементи матриці більшим чи меншим пріоритетом.

Для досягнення поставленої мети в дипломному проекті потрібно вирішити наступні задачі:

– вивчити теоретичні основи трасування та пошуку шляху;

– вивчити та систематизувати методи хвильового методу;

– розробити алгоритми функціонування програмного забезпечення системи трасування маршрутів на основі методів фронту хвилі;

– вибрати середовище розробки та мову програмування для розробки програмного забезпечення та обґрунтувати цей вибір;

– виконати програмування зазначеної програмної системи;

– здійснити тестування отриманого програмного забезпечення;

– за результатами вирішення попередніх задач, оформити пояснювальну записку та графічні матеріали дипломного проекту.

2.1.3 Загальні вимоги до розробки

Програмна система пошуку маршруту, що розробляється в межах даного дипломного проекту повинна забезпечувати набір функцій, необхідний для одержання у відповідність із параметрами користувача, зображення, яке

здатен вивести відповідний пристрій відображення з апаратним обмеженням кількості відтінків.

В розроблюваній системі необхідно реалізувати два методи поширення хвилі:

– ортогональний;

– діагонально-ортогональний.

Робота із системою повинна полягати в:

– створенню користувачем мапи розміром до 125 на 125 елементів для пошуку шляху;

– визначення типів елементів мапи, та вказання пріоритетності певних місць мапи;

– вибір швидкості відображення хвилі: покроковий чи швидкий метод;

– виборі масштабу мапи/

Розроблена система повинна забезпечувати виконання наступних функцій:

– введення параметрів розміру потрібної мапи, до 125 на 125;

– відображення результату введення на екрані в графічному виді;

– обробку мапи відповідно до обраного методу поширення хвилі;

– визначення різного рівню пріоритетності фрагментів мапи;

– визначення перешкод, старту та фінішу;

– збереження мапи для трасування;

– загрузку мапи для трасування.

До програмних продуктів, які розробляються у межах даного дипломного проекту пред’являються наступні вимоги: операційна система, яка підтримує функції розробленого програмного забезпечення – сімейство операційних систем Microsoft Windows; платформа – PC; вибір мови програмування та середовища розробки обґрунтовується в ході реалізації дипломного проекту.

2.2 Огляд аналогів розроблюваної програмної системи

На момент розробки програмної системи за для цієї дипломної роботи існує багато загальнодоступного програмного забезпечення для створення печатних плат. Отже розглянемо деякі із них. Також існує онлайн-система яка використовується як приклад пошуку шляху саме методом хвилі.

2.2.1 Sprint Layout

Це програмне забезпечення дозволяє вручну створити печатну плату малої та середньої складності. Ця система підтримує розміри плати до 300 мм на кожну сторону. У складі програми є бібліотека елементів, але інтегрована система трасування тільки допомагає прокладати провідники і не є автоматичним. Також мається можливість виводу результату у .PDF файл для подальшої печаті. Вартість ПО – 40 євро.

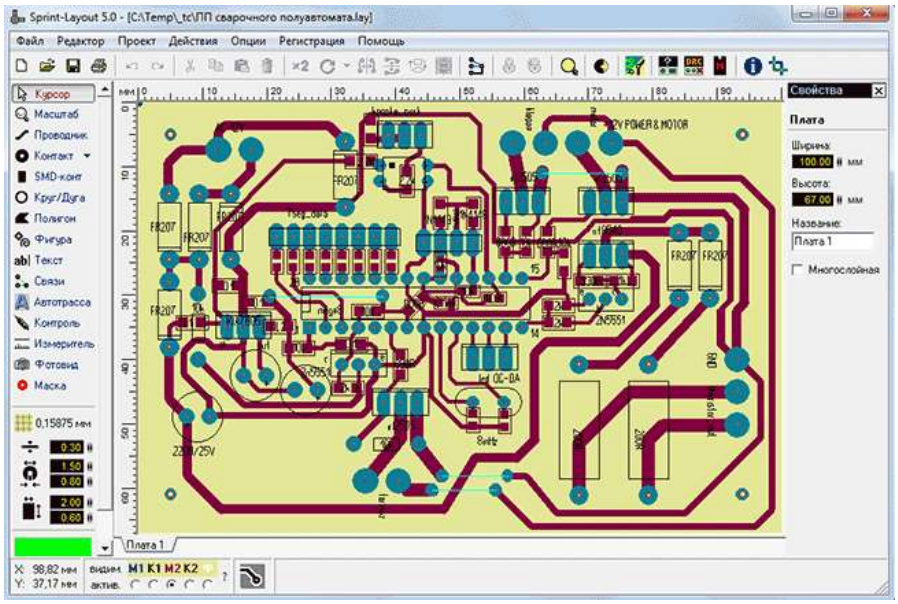


Рисунок 2.1 – Інтерфейс Sprint Layout

Отже це програмне забезпечення дозволяє тільки розмітити місця під елементи печатної плати, а вже потім пустити її на подальше виробництво. З цього аналога можна включити у розроблювану систему цільову низку складність результату, та зрозумілий інтерфейс, та додати автоматичне трасування доріжок провідників.

2.2.2 Layo1 PCB

Це програмне забезпечення дозволяє автоматично створювати печатну плату будь-якої складності. Інтегрована система трасування є автоматичною та дозволяє швидко прокладати шляхи. Також мається можливість зміни масштабу елементів плати до 0,1 мікрометрів. Підтримується до 16 шарів меді. Вартість ПО – 250 євро.

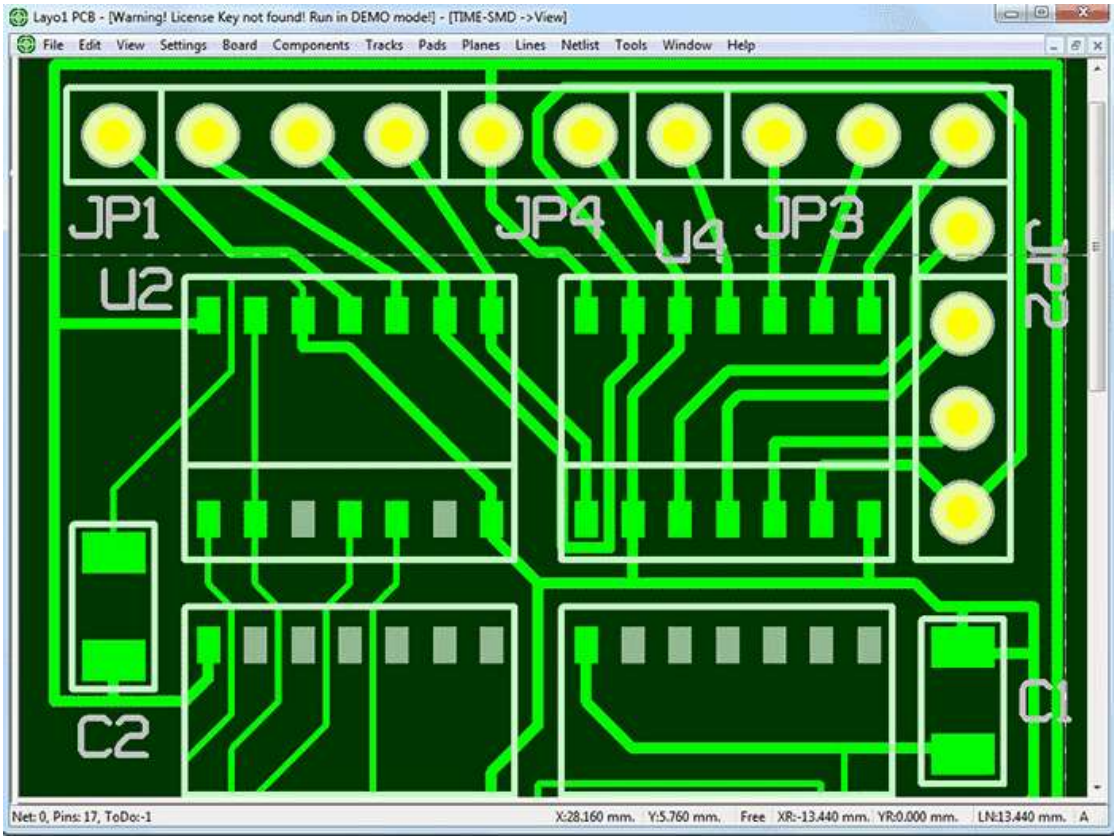


Рисунок 2.2 – Інтерфейс Layo1 PCB

Цей продукт програмної розробки зберігає мапу у форматі, який може використовувати тільки користувачі саме цієї системи. Для продукту дипломної роботи включимо зміну масштабу, та реалізуємо можливість перегляду результату навіть без розроблюваної системи.

2.2.3 «Хвильовий алгоритм Лі на JavaScript» із сайту upbyte.net

На відміну від попередніх аналогів цей – розроблений під браузери на мові програмування ECMAScript, також відомого як JavaScript. Також на відміну від аналогів ця система тільки демонструє роботу хвильового алгоритму і не дозволяє виробити свою мапу, а тільки отримати випадково генеровану мапу. Цю систему можна використовувати задля навчання новачків та ознайомлення із трасуванням шляху. У програмі відсутня можливість збереження результату.

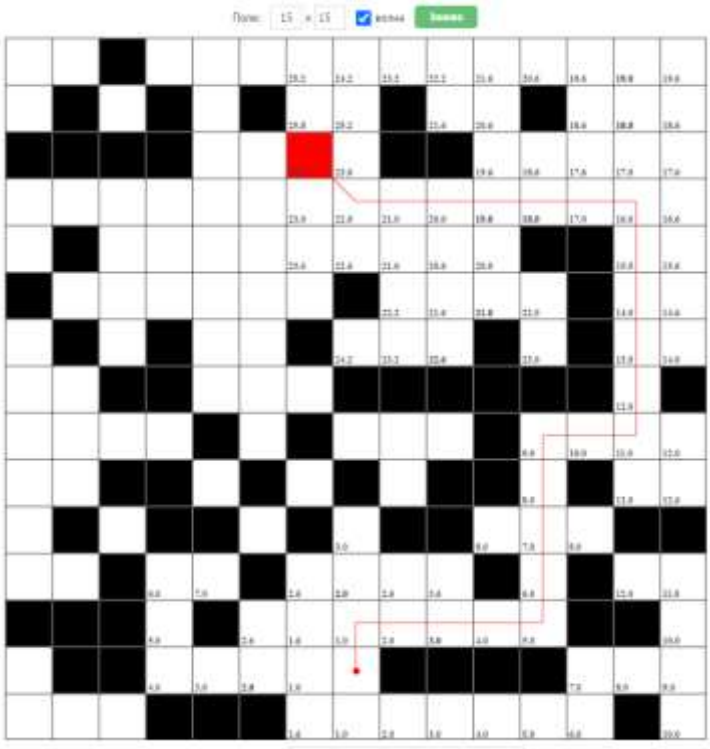


Рисунок 2.3 – Інтерфейс «Хвильовий алгоритм Лі на JavaScript» із сайту upbyte.net

Для розробки можна зазначити що мова програмування є однією із найпоширеніших, і може працювати майже на будь-якому пристрою. В якості основи для відображення пристосується HTML таблиця, також можна побачити індекси хвилі що допомагає зрозуміти принцип роботи алгоритму. Саме ці якості буде реалізовано у розроблюваної програмної системи.

2.3 Обґрунтування вибору технологій і програмних засобів реалізації проекту

Із розглянутих аналогів було зазначено деякі особливості, які буде реалізовано у розроблюваній системі. На основі цих особливостей було вибрано такий набір технологій:

По-перше мова програмування обрана JavaScript. Оскільки я маю досвід розробки програмного забезпечення на цій мові, а також виходячи з того, що один із аналогів було розроблено саме на цій мові, можна зробити висновок що цей вибір гарантує швидку розробку. Але із негативних сторін ECMAScript можна виділити відсутність типізації змінних, що може приводити до великої кількості помилок, які складно відстежити.

Найкращою та найпростішою парою технологій для візуалізації написаною JavaScript мовою програми є HTML та CSS. Я також маю досвід у роботі із цими технологіями. HTML разом із CSS дозволяє покращувати зовнішній вигляд програми не виходячи із браузеру.

Браузером для користуванням та тестуванням системи було вибрано Google Chrome. Цей браузер є одним із найпопулярніших, і найзручніший для розробки, оскільки вікно відладки має прямий доступ до усіх використаних для сторінки джерел, та моніторинг ресурсів пристрою користувача. Це дозволяє швидше та краще розробляти швидкодіючі програмні системи.

Для покращення загального виду буде застосовано іконки «foundation-icons» оскільки ліцензія дозволяє використовувати ці матеріали. Також, одним із вирішальних факторів є те, що на відміну від інших бібліотек іконок для HTML та CSS, не потребується постійне підключення до мережі Інтернет, оскільки є можливість збереження даних локально.

Як середу розроблення стаку технологій HTML, CSS та JavaScript було обрано WebStorm від компанії JetBrains. Цей програмний продукт професійного рівня розповсюджується для студентів безкоштовно, має

зручний інтерфейс та вбудовані файловий менеджер та підтримує технологію контроля версій Git.

Та саме Git було обрано для контроля версій та відстежуваня змін у проекті.

2.4 Аналіз сценаріїв використання розроблювальної системи

Оскільки основна мета цієї системи це мати можливість створювати та редагувати мапу незалежно від сфери використання, візуалізація та органи управління мають бути максимально нейтральними до кожної сфери.

Мова інтерфейсу обрана англійською, оскільки кількість та складність слів у інтерфейсі буде мінімальним, і увесь акцент має приходити саме на матрицю, яка є мапою. Також це збільшує кількість можливих користувачів цієї системи.

По-перше розміри мапи яку хоче створити користувач він буде вводити у стандартне браузерне вікно «prompt». Треба також обмежити користувача у спробах створити мапу із не цілим значенням ширини чи висоти, оскільки для кожного координату буде змальовано один елемент таблиці.

Для зазначення на карті перешкоди, або стінки було обрано коричневий колір, та іконку яку зазвичай використовують для меню. Ця іконка нагадує паркан.

Для зазначення на карті старту було обрано синій колір та іконку хатинки. Це символізує місце звідки користувач починає майже кожний маршрут за день. Саме з цього елементу буде починатися хвильовий алгоритм.

Фініш – має образ прапорцю червоного кольору. Червоний колір зазвичай протиставляють синьому. Оскільки для кращого пошуку шляху потребується застосування зворотного трасування, від фінішу буде також

застосовано алгоритм. Саме цьому фініш є водночас другим стартом і повною його протилежністю.

Фрагменти шляху із підвищеним пріоритетом, надалі буде називатися як прискорювач, оскільки якщо хвиля розповсюджується збільшенням індексу, то кращим шляхом буде той, де індекс хвилі найменший. Прискорювач повинен мати можливість бути відредагованим користувачем, отже коло вибору цього типу клітини має знаходитись регулятор впливу прискорювача на хвилю. Прискорювач буде зазначено як зелена іконка перемотування.

Фрагменти із пониженим пріоритетом, надалі буде називатися як сповільнювач, буде збільшувати індекс хвилі при проходженні через елемент цього типу. Це буде означати що шлях ймовірніше буде обходити такі елементи. Сповільнювач буде зображений як гори жовтого кольору.

Також має бути можливість змінювати масштаб мапи для покращення діапазону розмірів зручних для редагування.

Має бути також створено функціонал збереження та завантаження прогресу, отже потребуються відповідні кнопки.

Доповнити функціонал покликані перемикачі швидкості режиму, та направленості розповсюдження хвилі.

2.5 Висновки

В даному розділі дипломної роботи було обрано стек технологій, напрям розробки, сфера застосування, та приблизний зовнішній вигляд програмної системи. Було розглянуто та проаналізовано аналоги до потрібної системи. Також було визначено вимоги та завдання на розробку.

3 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

3.1 Розробка загальної структури програмного забезпечення

Виходячи із поставленої задачі, програма має мати такі логічні модулі:

1) модуль візуалізації;
2) модуль інтерфейсу;
3) модуль створення карти;
4) модуль валідації даних;
5) модуль введення розмірів карти;
6) модуль обробки прискорювачей;
7) модуль обробки сповільнювачей;
8) модуль поширення хвилі;
9) модуль повернення хвилі;
10) модуль загрузки json – файлу;
11) модуль формування json – файлу;
12) модуль збереження json – файлу;

Після того як було визначено логічних модулей треба розробити схему їх взаємодії.

Розглянемо модуль візуалізації. Через цей модуль користувач буде бачити матрицю елементів із іконками, які відображують різні типи елементів (стіни, старт, фініш, прискорювач, сповільнювач). Також цей модуль буде відображати індекси хвилі у процесі її поширення, та шлях у процесі зворотного трасування.

Розглянемо модуль інтерфейсу. Через цей модуль користувач буде використовувати такі модулі як модуль загрузки, модуль збереження та запускати сам алгоритм. У свою чергу на інтерфейс отримує данні про наявність старту та фінішу, данні щодо можливості трасування маршруту, та інші повідомлення.

Модуль створення карти буде отримувати від елементів інтерфейсу команди, які відповідають за визначення поточного режиму зміни елементів. У цьому модулі буде два циклу, які будуть створювати двомірний масив, якій далі передається до модулю візуалізації карти.

Модуль валідації даних – це той модуль, який не дозволяє створити два фінішних елемента, та не дозволяє створити карту розміром, який буде потребувати великого часу обчислення чи «це_точно_не_число». Потрібність у цьому модулю також обумовлена обраною мовою програмування, оскільки у Javascript відсутня типізація.

Модуль введення розмірів карти є модальним вікном, який отримує розміри по вертикалі та горизонталі від користувача. Ця інформація також валідується у відповідному модулі.

Модуль обробки прискорювачей як і сповільнювачей використовується через модуль поширення хвилі. Ці два модуля визначають поточні індекси, та додають їх до індексу хвилі при проходженні через елемент відповідного типу.

Модуль поширення хвилі отримує від модуля інтерфейсу команду, а від пам’яті програми – координати старту та масив елементів із типами цих елементів. Він оброблює кожен елемент сусідній до стартової точки, передаючи до нього індекс хвилі. Якщо елемент по типу відноситься до прискорювачей, чи сповільнювачей – цей елемент обробляється у відповідних модулях, та результат повертається назад. Якщо сусідній елемент по типу відноситься до фінішу – то модуль припиняє свою дію.

Модуль повернення хвилі реалізує алгоритм оберненого трасування, а для цього йому потрібно отримати із пам’яті програми координати фінішу та масив елементів. Він знаходить сусідній елемент до фінішної із найменшим індексом хвилі, передаючи до нього іконку стрілки, яка вказує шлях від старту до фінішу.

Усі останні модулі позволяють використовувати заповнену карту знову, зберігаючи на пристрої користувача JSON, у який вказується координати

старту, фінішу та масив елементів, уникаючи елементи знайденого шляху. Таким чином ми можемо знаходити різні маршрути на старих картах, змінюючи їх після знайденого шляху.

Приблизний вигляд структури можна побачити на рисунку 3.1.

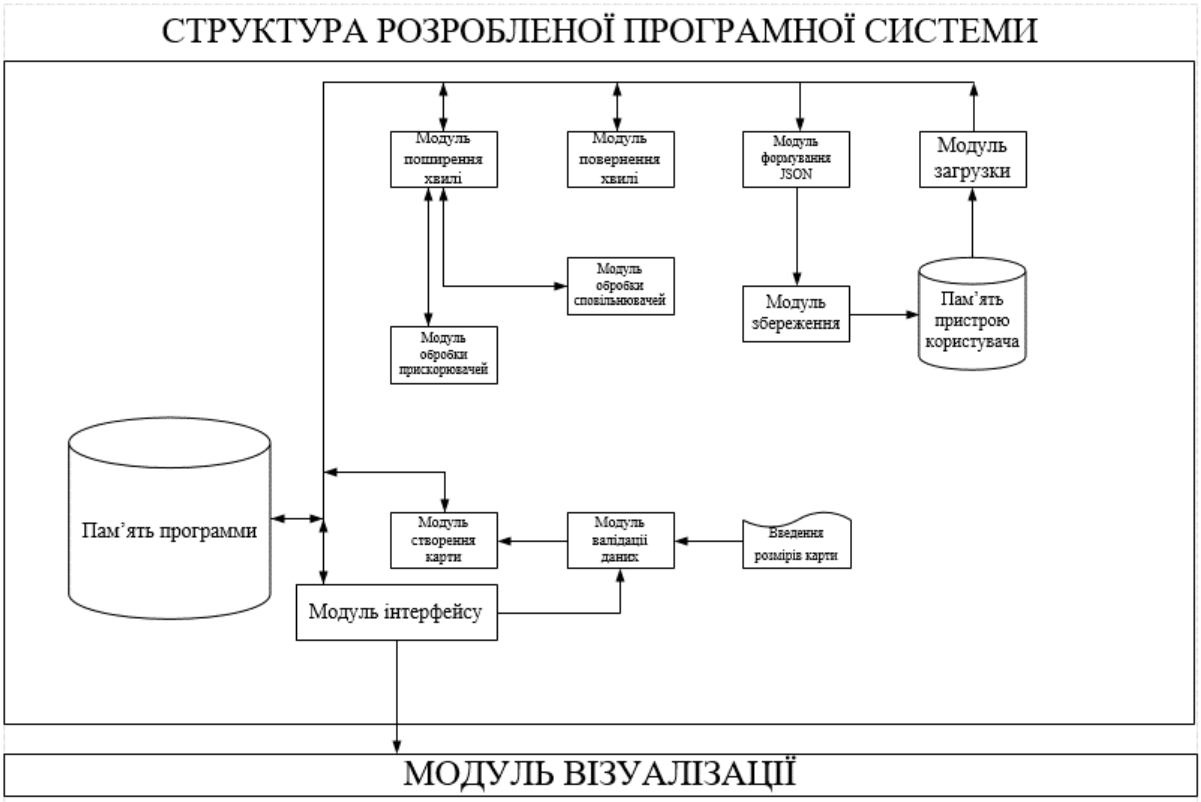


Рисунок 3.1 – Структура розробленої програмної системи

3.2 Розробка інтерфейсу користувача

Для візуалізації програми було використано HTML із елементами CSS. HTML у цій схемі використовується для розмітки сторінки та для створення елементів інтерфейсу. CSS було використано для надання належного стилю. А тепер розглянемо як було реалізовано інтерфейс.

По-перше програмна система має три контейнера для карти, селектору режиму редагування і інтерфейс трасування.

У першому контейнері ми маємо таблицю, яка буде відображати карту, меню редагування масштабу, меню збереження, створення та експорту карти.

Таблиця має елементи, які змінюють свій колір та зовнішній вигляд в залежності від режиму, індексу хвилі та наявності шляху який пролягає через цей елемент. Усі елементи сторінки мають однакові параметри розміру, чорну рамку в 1 піксель, відстань до внутрішніх даних є також однаковою та становить 5 пікселів.

Другий контейнер має у собі радіо кнопки, із надписами та іконками які відображують той тип елементу який вмикається при вибору одного з даних режимів. Режими прискорювача та сповільнювача також мають модулі регулювання потужності, для цього було використано класичні html input-и, які були налаштовані на діапазон 0.01 – 0.9 із кроком у 0.01, початкове значення дорівнює 0.5. Ці інпути мають тип number що не дозволяє користувачеві ввести туди щось окрім цифр.

У третьому контейнері знаходяться кнопки запуску алгоритму. Також чекбокси ортоганально-діаганального режиму та режиму крок-за-кроком із кнопкою, що вмикає наступний крок.

За допомогою CSS усім контейнерам було надано чорну рамку у 1 піксель та відступи до краю елементу у 5 пікселів та від краю елементу до іншого 5 пікселів. Таким чином елементи інтерфейсу логічно поділені на розділи. Також за допомогою CSS інпути мають сталу ширину у розмірі 45 пікселів, яка не дає їм розповсюджуватись в ширину, що псує загальний вид.

3.3 Розробка схем алгоритмів

Виходячи із описаних вище структури та інтерфейсу можна зазначити основні алгоритми програми, а саме алгоритм поширення та повернення хвилі.

Алгоритм поширення хвилі повинен починатися з того, що навколо зазначеної точки старту розповсюджується хвиля, що надає усім сусідам (в залежності чи включено ортоганально-діагональний режим) цієї точки, які можуть містити шлях, індекс хвилі 1, а якщо ці сусідні елементи по типу

дорівнюють прискорювачу, чи сповільнювачу то індекс змінюється відповідно до типу та значенні бонусу, або пенальті відповідного елементу. Після чого візуалізація масиву оновлюється, якщо режим «крок за кроком» було включено. Усі сусіди, що отримали індекс потрапляють до масиву nextElements, елементи якого потім передаються до черги elementsQueue. Ця черга працює на алгоритмі FIFO. Кожен крок цієї черги наповнює масив nextElements сусідами, подібно до першої функції цього алгоритму, але додаючи до індексу хвилі 1 на кожному повторі черги. Таким чином ми отримуємо цикл, який продовжується доки уся карта не буде заповнена індексами хвилі. Залишилось тільки добавити розрив у випадку знайдення фінішу, що захистить алгоритм від зайвих дій. Після того як фініш було успішно знайдено залишається тільки оновити візуалізацію. Виходячи із описаного функціоналу маємо схему на рисунку 3.2.

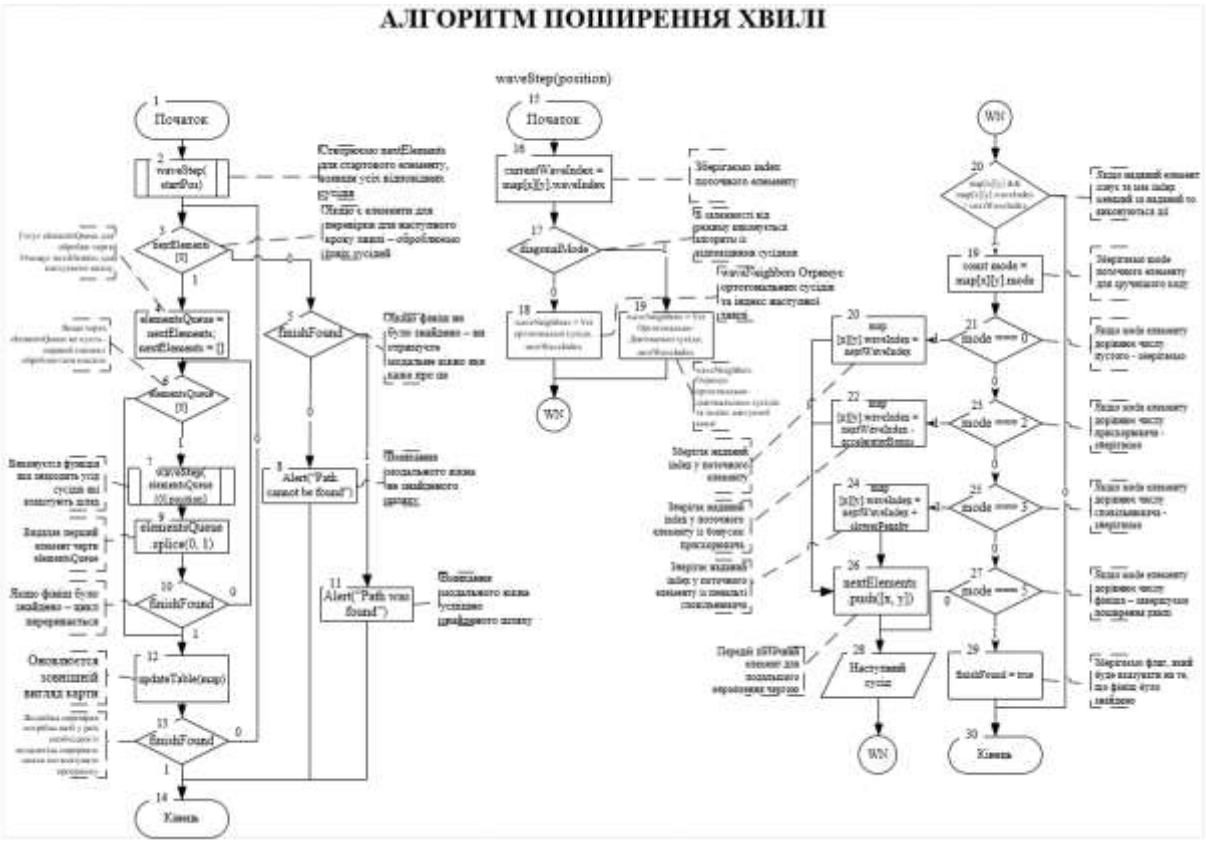


Рисунок 3.2 – Алгоритм поширення хвилі

Алгоритм повернення хвилі починається з того, що навколо зазначеної точки фінішу визначається сусідній (залежно від ортоганально-діагонального

режиму) елемент із найменшим індексом хвилі. Після цього, якщо режим «крок-за-кроком» активовано, оновлюється візуалізація карти. Таким чином визначається напрям найкоротшого шляху до фінішу. Але цей напрям потрібно продовжити до самого старту. Ось чому далі, взявши за середину попередній сусідній елемент ми шукаємо його «найменшого» сусіда. І так далі, поки не прокладемо шлях до старту. Після чого оновлюємо візуалізацію, та показуємо вікно із повідомленням «шлях було знайдено». У наслідку із описаного функціоналу маємо схему на рис 3.3.

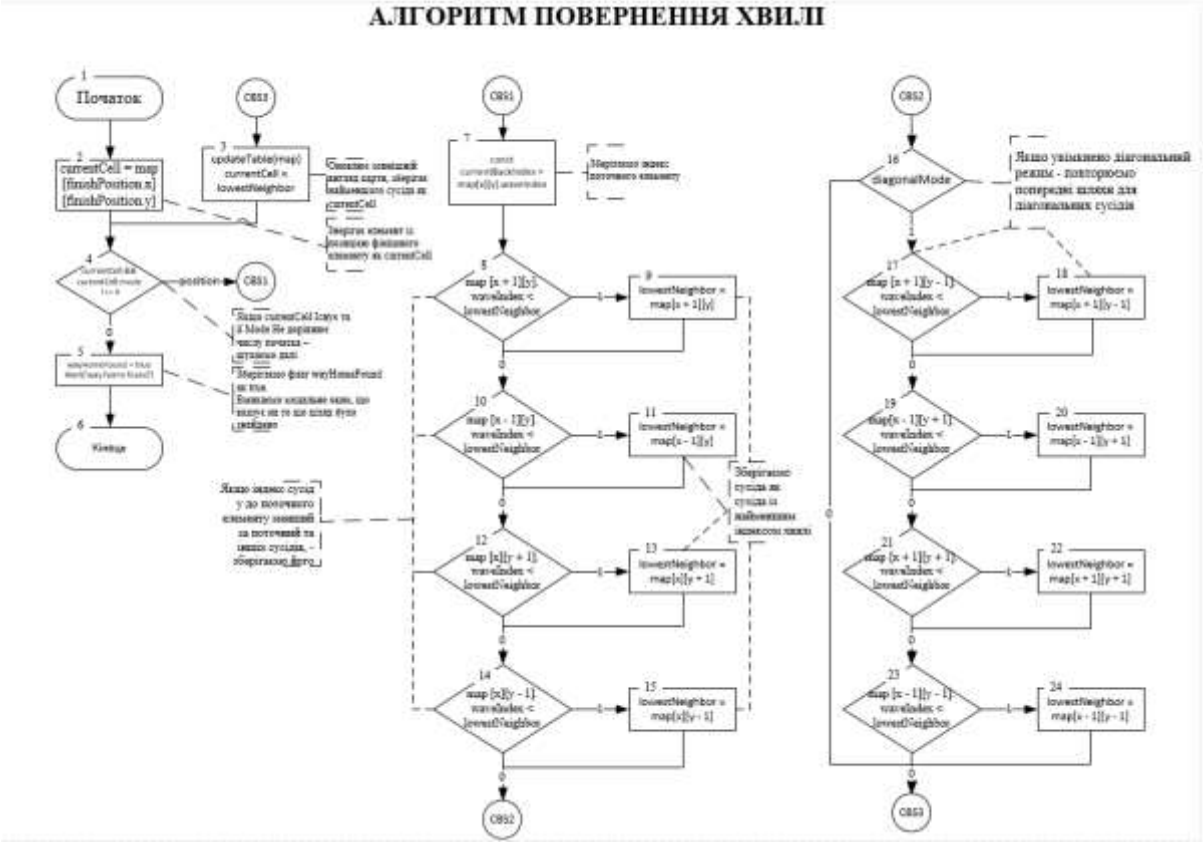


Рисунок 3.3 – Алгоритм зворотного трасування

3.4 Розробка програмної системи

3.4.1 Функія поєднення інтерфейсу із скриптом

Інтерфейс програми має велику кількість елементів що мають поєднатися із скриптом, тому цей розділ має декілька розділених функцій. Усе починається із створення глобальних змінних, а саме: mod, modsContent,

config, startPosition, finishPosition, map, currentMod, acceleratorBonus, slowerPenalty, stepMode, diagonalMode, currentScale, currentCellStyle, table. Ці змінні можуть потребуватись у різних функціях на різних етапах програми, саме тому я вирішив зробити їх глобальними.

Далі йде функція що має назву «initModSelector». Ця функція не потребує ніяких вхідних даних. У своїй суті ця функція робить наступне: перебирає масив кнопок які відповідають за вибір режиму редагування таблиці і надає їм onChange функцію що змінює currentMod на той, що відповідає надпису коло кнопки. Цей масив отримується завдяки пошуку у document по name ‘mod’. Далі із документу функція знаходить інпут за id ‘accelerator’, та надає його onChange функцію, яка змінює acceleratorBonus відповідно до введеного до цього інпуту значення. Далі до еленту із id ‘slower’ дається onChange функція, яка оновлює slowerPenalty згідно до введеного до цього інпуту значення.

Наступна функція – «initStarter», яка поєднує HTML елементи із контейнеру для інтерфейсу трасування. Спочатку із документа знаходять елементи по id ‘start’. Цьому елементу надається функція onClick, яка перевіряє чи було визначено координати старту та фінішу, та якщо користувач не зробив цього - видає відповідне повідомлення. Якщо можна починати трасування то блокуються чекбокси, які відповідають за режими «крок-за-кроком» та «діагонально-ортогональний». Потім блокуються інпути які визначають acceleratorBonus та slowerPenalty. Після цього перевірка на режим «крок-за-кроком» визначає яку функцію трасування запускати, покрокову, чи звичайну. Якщо було вибрано пошаговий режим то кнопка «Next Step» стає активною, а кнопки вибору режиму редагування карти блокуються. Запускається функція, яка була назначена на onClick елементу, який знайшли у document завдяки id ‘next’, а саме – кнопку «Next Step». Ця функція виконує функцію nextStep, для якої не потрібно передавати змінних.

Далі йде функція «initScaleMenu». Ця функція не потребує вхідних змінних. Спочатку із document знаходять елемент за id «plus». Цей елемент – кнопка збільшення масштабу. onClick цій кнопці дається наступний: до currentScale додається один, потім, якщо currentScale стає більшим чи дорівнює 20 ця кнопка стає неактивною, а якщо currentScale набуває значення більше за один – кнопка мінусу стає активною. Після цього виконується функція що оновлює візуалізацію графу відповідно до нового масштабу. Далі знаходиться по id «minus» із document. Це кнопка для зменшення масштабу, функція onClick привласнюєтеся дуже схожа на функцию плюса onClick. А саме: currentScale віднімається один, якщо currentScale менший за 20 то кнопка плюса стає активним, а якщо менше чи дорівнює один – кнопка мінуса стає неактивною. Зрештою оновлюється масштаб візуалізації графу. Функція оновлення масштабу робить наступне: оновлює параметри currentCellStyle за формулами (див. рис 3.4) виходячи із currentScale, та виконує функцію що малює нову HTML таблицю, виходячи із нових параметрів стилю текста.

```
function applyScale() {
    currentCellStyle.width = 30 * currentScale / 10 + 'px';
    currentCellStyle.height = 30 * currentScale / 10 + 'px';
    currentCellStyle.fontSize = currentScale * 10 + '%';
    currentCellStyle.iconSize = 18 * currentScale / 10 + 'px';

    updateTable(map);
}
```

Рисунок 3.4 – Функція оновлення масштабу

Остання функція цього розділу має назву «initSaveMenu». У цій функції із document буде знайдено за id елементи «newMap», «loadMapImport», «loadMap», «saveMap». Усі ці елементи є HTML кнопками. Кнопка newMap має назначену на onClick функцію, що визиває для x та y функцію яка вказує користувачу ввести ці координати, маючи за стандартну відповідь 7. Функція повторює запит до користувача поки користувачем не буде введено число що проходить ці умови: не менше чи дорівнює нулю, не число із плаваючий точкою, не більше за 126. Після вводу обох координат створюється config, що

має mapSize, де параметри x та y дорівнюють введеним числам. Після цього кнопки newMap, saveMap/ loadMapImport стають неактивними, а loadMap – активною. Після цього виконується init, у який передається true, що означає що потрібно визвати initMap для нової карти, потім карта оновлюється. Кнопка loadMapImport є допоміжною до інтерфейсу загрузки JSON, та виконує тільки роль вибору файлу до загрузки. Загрузка JSON виконується у функції onClick кнопки loadMap. По кліку буде створено файл files, який дорівнює файлам, що були вибрані у loadMapImport. Далі, якщо довжина цього масиву менша або дорівнює нулю – функція припиняється, та користувач отримує повідомлення, що спершу треба вибрати файли. Далі створюється FileReader, котрому на функцію onLoad назначається наступне: парсинг отриманого результату, перевірка чи має цей результат config та savedMap, далі до програмної системи передаються параметри, що були у файлі, а саме: карту із елементами та іконками, позиції старту та фінішу, але якщо config та savedMap не існує, то користувач отримує повідомлення що із файлом «щось не так». Далі інтерфейс набуває стану як після створення нової карти. Виконується функція init, до якого передається false, що означає що map вже існує та не потребує створення по координатам, після чого малюється карта. І останньою кнопкою є saveMap, яка по кліку створює новий масив із елементів поточного map, де перебором по x та y кожному елменту затирається waveIndex. Далі формується JSON, який має отримує нову карту, розміри карти, позиції старту та фінішу. Далі створюється якір, до якого даються параметри для скачування файлу, та оскільки цей якір не додається на сторінку то відразу виконується клік по цьому якорю.

3.4.2 Функція відображення карти

Ця функція є посередником між скриптами інтерфейса та хвильовими алгоритмами. Також ця функція інколи має назву оновлення карти чи таблиці.

Коли функція відображення отримує карту для відображення, перш за все із конейнеру карти видаляється стара таблиця, якщо її вдалося знайти, потім створюється нова таблиця, яка отримує id «table».

Коли до функції повертається нова таблиця два циклу починають створювати рядки та стовпи, які наповнює елементами, стиль яких визначає багато параметрів. Конфіг масштабу визначає висоту та ширину клітин, розмір шрифту та іконок, які вказують на тип елементу. У кожного типу елементу є id, який може набувати такі параметри: «empty», «wall», «accelerator», «slower», «start», «finish», «path». Далі, якщо у елементу немає параметру content, то виконується функція оновлення елементу, яка ілюструє індекс хвилі шрифтом чорного кольору. Інакше – якщо у елементі немає waveIndex, то застосовується updateCell до якого передається цей елемент та пустий рядок, та додається іконка елемента. Інакше – якщо у елемента є і data і у даті arrow – оновлюється елемент на стрілку кольору сповільнювача чи прискорювача відповідно. Інакше – елемент отримує колір який було вибрано для прискорювача та сповільнювача та відповідну іконку. Інакше – просто елемент просто отримує індекс хвилі чорним кольором на білому фоні.

Після чого на цей елемент таблиці дається на клік функція що запобігає встановленню двох стартів та фінішів, зберігає позицію елементу як старт чи фініш, у разі успішного встановлення його, або видаляє ці данні якщо по фінішу чи старту натиснути із режимом «empty». Зберігається поточний режим редагування як id клітини.

3.4.3 Функції трасування маршруту, та оберненого трасування

Цей розділ функцій буде виконуватися тільки у разі кліку по відповідній кнопці, та якщо карта до цього готова, тобто мається і старт, і фініш. Оскільки у програмній системі обчислення та робота алгоритму міститься в окремому блоці – є можливість змінювати інтерфейс та стиль відображення без додаткових зусиль на адаптацію алгоритму.

Для цього розділу спочатку створюються наступні змінні: «elementsQueue», «nextElements», «biggestWaveIndex», «finishFound», «wayHomeFound», «currentCell». Усі ці змінні отримують значення пустого масиву, 0, false, або null відповідно. Це було зроблено щоб ці змінні були доступні у всіх наступних функціях. Для режиму «крок-за-кроком» також створюються waveStarted та pathFinderStarted.

Далі йде розділення алгоритмів на покроковий та швидкий режими. Деякі функції застосовуються в обох варіантах, що заощаджує час, який потребується для прочитання алгоритму. Також, завдяки цьому програмна система може бути удосконалена та доповнена новим функціоналом. Саме таким чином вийшло додати блоки сповільнення та прискорювання, що збільшило можливі випадки застосування.

Перша функція поширення хвилі – застосовується для швидкого режиму. Ця функція виконується після кліку по кнопці «Start» та має назву «waveStart». До цій функції не передається ніяких змінних, вона сама використовує глобальну змінну map, тобто карту елементів. По-перше виконується функція «checkStarEndReady»[1], яка перевіряє чи готова карта для поширення хвилі. Далі – виконується функція «waveStep»[2], до якої передається позиція із елементом «старт». Після цього «nextElements» стає масивом елементів що, згідно із ортогональним чи діагонально-ортогональним режимом, є сусідами до стартового елементу. Далі опрацьовується цикл, який виконується доки існує перший елемент масиву «nextElements». У цьому циклі виконується наступне: elementsQueue отримує

значення масиву nextElements, далі у свою чергу nextElemets стає пустим масивом. Таким чином ми створюємо чергу яка буде наповнювати nextElemtns сусідами сусідів, при цьому не втручаючи нові доступні для поширення хвилі елементи. Далі – наступний цикл, який оброблює чергу, а саме: виконує «waveStep»[2] до першого елементу elementsQueue, після чого цей елемент видаляється із черги. Цей цикл продовжує свою роботу доки існує елементи черги, чи доки фініш не було знайдено. Далі перший цикл цієї функції оновлює візуалізацію карти. А якщо фініш було знайдено, тобто змінна finishFound дорівнює true, то цей цикл переривається і функція waveStart продовжує своє виконання. Якщо фініш було знайдено – виконується функція «findWayHome», а інакше – алгоритм перестає опрацьовуватись, а користувач отримує повідомлення що фініш не було знайдено, але цю карту ще можна зберегти щоб відредагувати після оновлення сторінки.

Функція «findWayHome» виконує зворотне трасування, тобто вказує шлях знаходячи його від фінішу до старту. Ця функція по-перше надає змінній currentCell елемент карти із координатами finishPosition, тобто елемент «фініш». Далі виконується цикл який працює доки існує currentCell та його mode не дорівнює чотирьом, тобто якщо currentCell не є фінішним елементом. В тілі циклу currentCell набуває значення, яке повертає функція «comeBackStep»[3], до якої передається currentCell.position, тобто позиція елементу карти currentCell, та оновлюється візуалізація. Після циклу значення wayHomeFound набуває true, що означає що шлях було знайдено та зображено для користувача. Кнопка startButton стає неактивною.

Друга функція поширення хвилі застосовується для покроковому режиму. Ця функція виконується при натисканні «Start», коли режим є «крок-за-кроком», і має назву nextStep. Але після першого тиску по цій кнопці вона стає неактивною, і для подальшої роботи алгоритму потребується натискати кнопку NextStep на кожній ступені алгоритму. Функція «nextStep» перш за все перевіряє чи не було знайдено хвилею фініш, але якщо

«checkStarEndReady»[1] повертає false, тобто карта не готова для трасування, то алгоритм не виконується, а користувач отримує відповідне повідомлення. Далі йде перевірка чи було знайдено хвилею фініш. Якщо ні – йде перевірка чи функція startWaveStep вже почала свою роботу, тобто якщо флаг waveStarted дорівнює значенню true. Якщо waveStarted дорівнює значенню false, тобто startWaveStep ще не почала свою роботу, – вона починається. Інакше – виконується функція continueWave. Якщо значення finishFound дорівнює true, тобто хвиля дійшла до фінішу, то йде перевірка чи вже було розпочато алгоритм зворотного трасування, тобто якщо pathFinderStarted дорівнює true. Якщо твердження позитивне – виконується функція findWayHomeStep, а якщо негативне – то спочатку currentCell отримує елемент карти із координатами finishPosition, тобто елемент «фініш», pathFinderStarted стає true, і вже потім виконується функція findWayHomeStep.

Функція startWaveStep виконує функцію «waveStep»[2], до якої передає startPosition, тобто позицію старту хвилі. Після цього waveStarted набуває значення true, та оновлюється візуалізація карти.

Функція continueWaveStep виконується після функції startWaveStep, коли «nextElements» є масивом елементів що, згідно із ортогональним чи діагонально-ортогональним режимом, є сусідами до стартового елементу. Далі перевіряється чи існує перший елемент масиву «nextElements». Після чого виконується наступне: elementsQueue отримує значення масиву nextElements, далі у свою чергу nextElemets стає пустим масивом. Таким чином ми створюємо чергу яка буде наповнювати nextElemtns сусідами сусідів, при цьому не втручаючи нові доступні для поширення хвилі елементи. Далі – виконується цикл, який оброблює чергу, а саме: виконує «waveStep»[2] до першого елементу elementsQueue, після чого цей елемент видаляється із черги. Цей цикл продовжує свою роботу доки існує елементи черги, чи доки фініш не було знайдено. Після цього оновлюється візуалізація карти. А якщо фініш було знайдено, тобто змінна finishFound дорівнює true, то nextElements набуває

значення пустого масиву. Якщо не існує першого елементу масиву «nextElements», тобто масив пустий, і фініш не було знайдено, тобто finishFound дорівнює false, користувач отримує повідомлення що фініш не було знайдено, але цю карту ще можна зберегти щоб відредагувати після оновлення сторінки. Після чого wayHomeFound набуває значення true, що буде означати що при наступному натисканні по кнопці «Next Step» користувач отримує повідомлення, що потрібно оновити сторінку щоб спробувати спочатку.

Функція findWayHomeStep виконує зворотне трасування, тобто вказує шлях знаходячи його від фінішу до старту. Ця функція перевіряє чи існує currentCell та чи його mode не дорівнює чотирьом, тобто якщо currentCell не є фінішним елементом. Якщо так – то currentCell набуває значення, яке повертає функція «comeBackStep»[3], до якої передається currentCell.position, тобто позиція елементу карти currentCell, та оновлюється візуалізація. Якщо currentCell не існує, його mode не дорівнює чотирьом, значення wayHomeFound набуває true, що означає що шлях було знайдено та зображено для користувача. Також кнопка startButton стає неактивною.

Функція [1] «checkStarEndReady» пристосовується для перевірки карти на наявність старту та фінішу, та для повідомлення користувача о необхідності спершу розмістити їх. По-перше йде перевірка на наявність startPosition, тобто чи було зазначено елемент із типом «старт». Якщо startPosition не існує – то користувач отримує повідомлення що треба його встановити, а функція повертає false. Далі, якщо startPosition існує – перевіряється наявність finishPosition, тобто чи було зазначено елемент із типом «фініш». Якщо finishPosition не існує – то користувач отримує повідомлення що треба його встановити, а функція повертає false. А якщо і startPosition, і finishPosition існують – функція повертає true.

Функція [2] «waveStep» пристосовується для розповсюдження хвилі. Спершу якщо у переданого до функції елементу немає індексу хвилі – елемент

отримує індекс 0. Також зберігається індекс поточного елементу. Далі у залежності від режиму усі ортогональні чи ортогонально-діагональні сусіди передаються по одному до функції «waveNeighbor». Ця функція перевіряє наявність взагалі елементу із такими координатами, далі, якщо індекс хвилі у цьому елементу є менший за той, що може бути до нього вписаний – вписується новий індекс, відповідно до режиму поточного сусіда. Індекс хвилі стандартно збільшується на один при ортогональному сусідстві, чи на 1.4, якщо сусід є діагональним. Цей випадок використовується для елементів із типом пустого. Якщо режим сусіда це сповільнювач – до стандартного збільшеного індексу хвилі додається slowerPenalty, який визначає користувач. А якщо режим сусіда це прискорювач – від стандартного збільшеного індексу хвилі віднімається acceleratorBonus, який визначає користувач. Після зміни індексу – цей сусід додається до глобального масиву nextElements. Але якщо було знайдено фініш – обробка черги сусідів припиняється, та finishFound стає true.

Функція [3] «comeBackStep» повертає сусідній елемент із найменшим індексом хвилі до переданого до функції елементу. Також ця функція записує у елемент данні які потрібні щоб у цьому елементі була стрілка шляху, яка буде вказувати напрям шляху. По-перше у цій функції створюється конфіг id по яким буде обрано іконку для елементу. Також створюються тимчасові змінні neighborIndex та lowestNeighbor. Ще зберігається індекс хвилі із поточного елементу, щоб не потрібно було звертатися кожного разу до карти. Далі у залежності від режиму усі ортогональні чи ортогонально-діагональні перевіряються існування в них індексу хвилі, та чи не є цей індекс менший за найменший із перевірених, після чого цей сусід зберігається як найменший. Після того як усі сусіди було перевірено визначається чи не є цей сусід фінішом, і якщо це не фініш то до цього елементу записується яка саме стрілка повинна бути зображена, а у data передається що цей елемент має стрілку.

3.5 Тестування програмної системи

Для того, щоб перевірити, чи виконує програмний продукт функціональні вимоги, потрібно виконати описані нижче дії.

Спочатку перевіримо валідацію полів вводу ширини та висоти мапи. Для цього запустимо файл index.html з папки ПО. Це продемонстровано на рисунку 3.4.

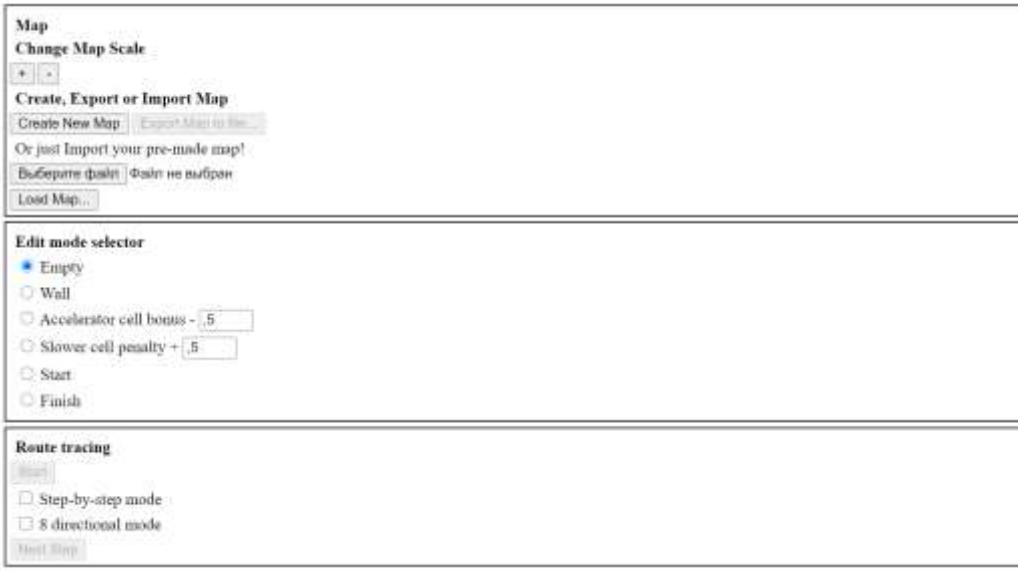


Рисунок 3.4 – Початкове вікно програмної системи

Після чого, натиснемо на кнопку “Create new map” (рис.3.5) для створення нової мапи. Як запропоноване число маємо сім.

Рисунок 3.5 – Вікно для вводу даних

Виходячи із технічного завдання значення для висоти та ширини мапи повинні бути тільки у числовому проміжку значень від 1 до 125 включно.

Також, програма приймає тільки цілі значення (рис. 3.6). Перевіримо граничні значення 0 та 126:

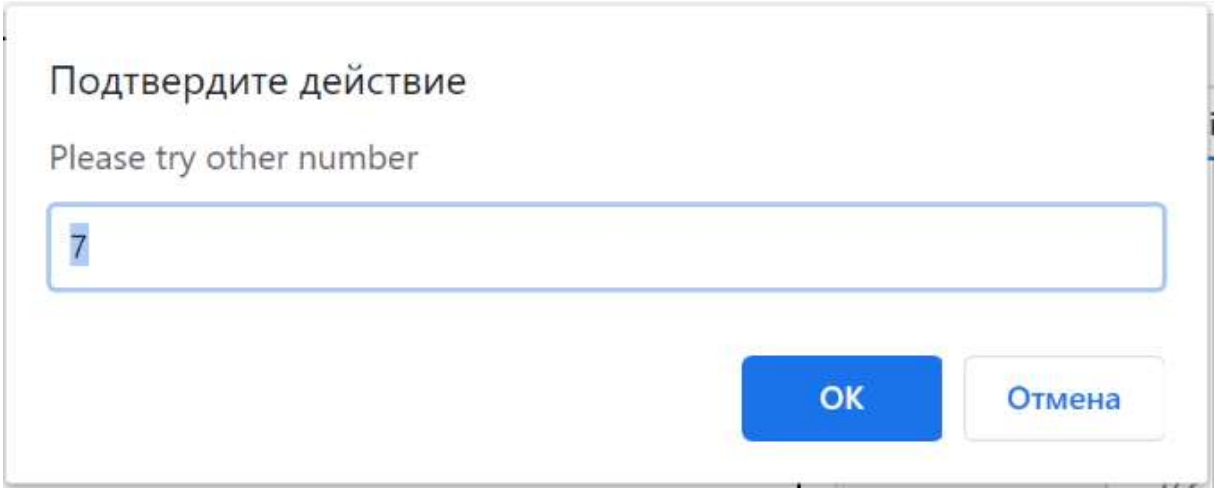


Рисунок 3.6 – Вікно з після вводу невірних значень

Якщо ввести данні які не задовольняють перевірку у вікні користувач отримує повідомлення «Please try other number» - (Будь ласка спробуйте інше число).

Також, ґрунтуючись на технічному завданні функціонал системи має дозволяти загрузити вже готову мапу для пошуку шляху, для цього натиснемо на кнопку «Выберите файл…» та знайдемо файл формату .json. Перевіримо, що мапа функціональна – редагуючи перешкоди та елементи старт/фініш. Для цього після натискання на кнопку «Load Map» та функціоналом розділом «Edit mode selector» (рис. 3.7), а саме кнопками: «Empty», «Wall», «Start», «Finish» можна обрати тип, на який змінюється обраний елемент мапи.

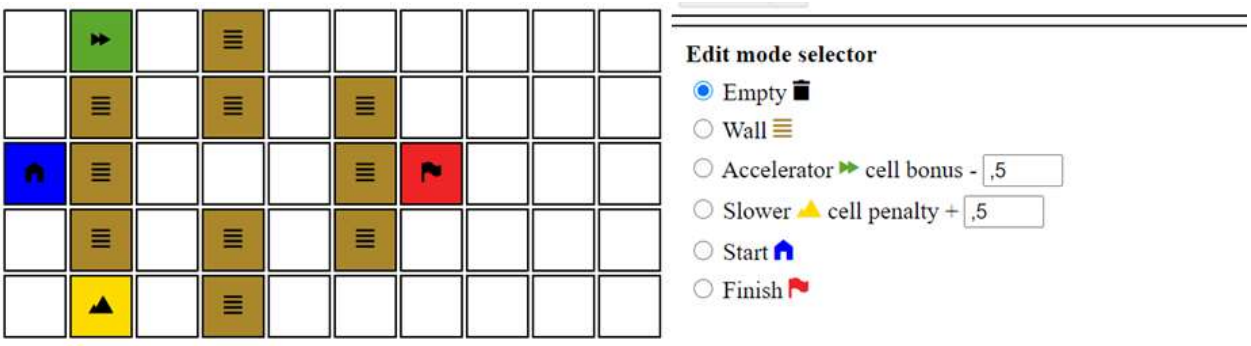


Рисунок 3.7 – Редагування готової мапи

Тестування режиму трасування маршруту у режимі «Step by step» в секторі «Route tracing»:

Якщо режим «Step by step» (рис 3.8) активовано, то після натиснення кнопки «Start» кнопка «Next Step» стає доступною для натиску. За її допомогою можливо прослідити трасування на кожному кроці.

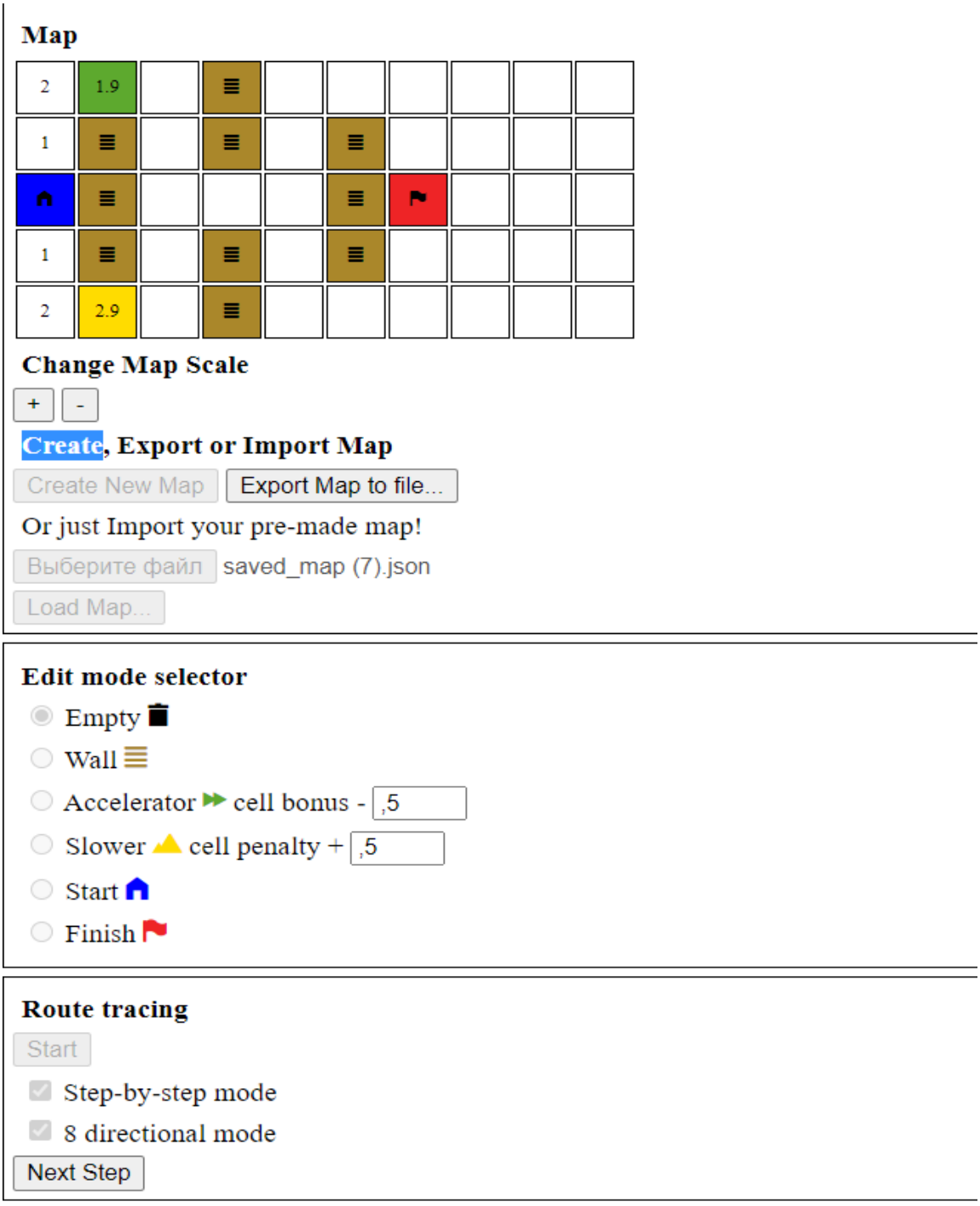


Рисунок 3.8 – Режим покрокового трасування

Тестування меню масштабу карти. За допомогою кнопок «+» / «-» біля підпису «Change Map Scale» карта має змінювати свій розмір. Перевірку

виконано на мапі розміром десять на десять. При мінімальному значенні масштабу маємо (рис. 3.9).

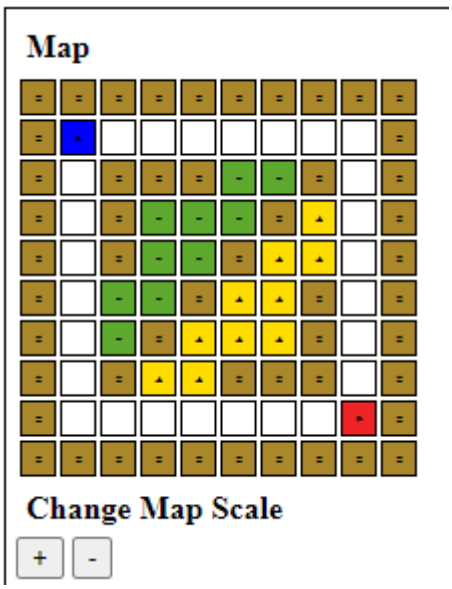


Рисунок 3.9 – Мапа із мінімальним масштабом

А при максимальному значенні масштабу маємо (рис. 3.10).

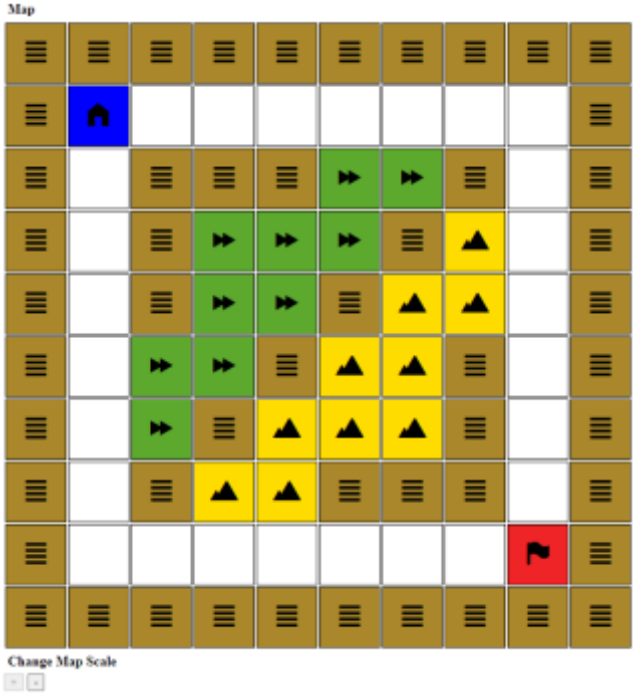


Рисунок 3.10 – Мапа із максимальним масштабом

Отже, завдяки зміні масштабу система може бути для використання як малих мап, так і для великих.

За технічним завданням, після завершення трасування – всі кнопки мають стати неактивними, окрім кнопки «Export Map to file» (рис. 3.11).

Натиснувши на яку стає можливо зберегти карту в форматі .json на свій пристрій.

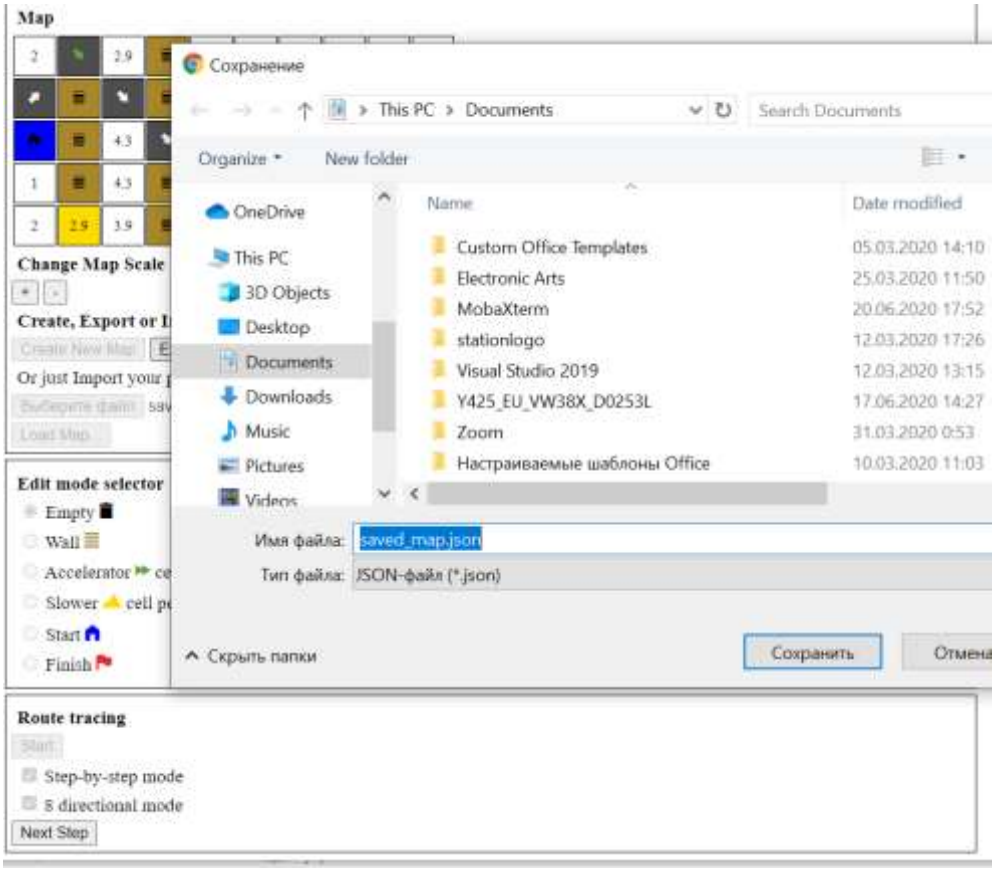


Рисунок 3.11 – Збереження мапи

Для тестування блоків прискорення та сповільнювання використаємо збережену карту. Виходячи із технічного завдання прискорювачи повинні зменшувати вартість хвилі на заданий користувачем індекс (у поточному прикладі – 0.5), а сповільнювач навпаки – збільшувати на індекс вказаний користувачем (у поточному прикладі – 0.5), як висновок маршрут має пролягати саме через прискорювач, а не сповільнювач. У результаті тесту маємо таку відповідь програмної системи як зображено на рисунку 3.12.

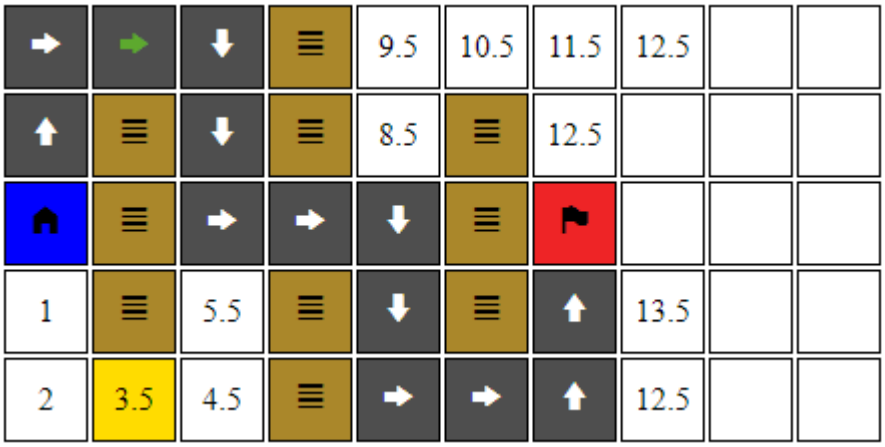


Рисунок 3.12 – Тестування блоків прискорення та сповільнення

Отже програма відпрацьовує ці елементи саме як потребує завдання. Також, на елементі із типом прискорювача, через який пролягає шлях, стрілка шляху є зеленого кольору, вказуючи що на цьому місці був прискорювач.

Для тестування ортогонально-діагонального режиму було обрано таку мапу, що зображена на рисунку 3.13.

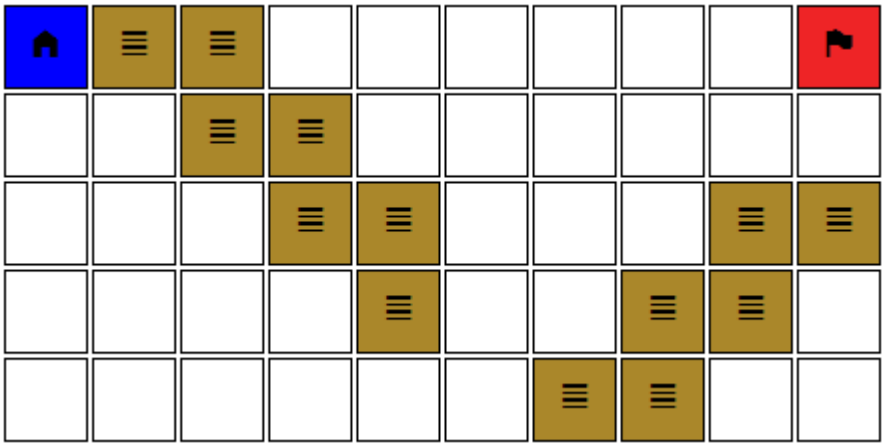


Рисунок 3.13 – Мапа для тестування діагонального режиму

Виходячи із технічного завдання увімкнувши діагональний режим шлях на цій мапі має пролягати уздовж елементів зазначених стінками. Індекс хвилі при поширенні через діагонально сусідній елемент має збільшуватись на 1.4 замість одного. Без діагонального режиму система повертає такий шлях, зо показан на рисунку 3.14.

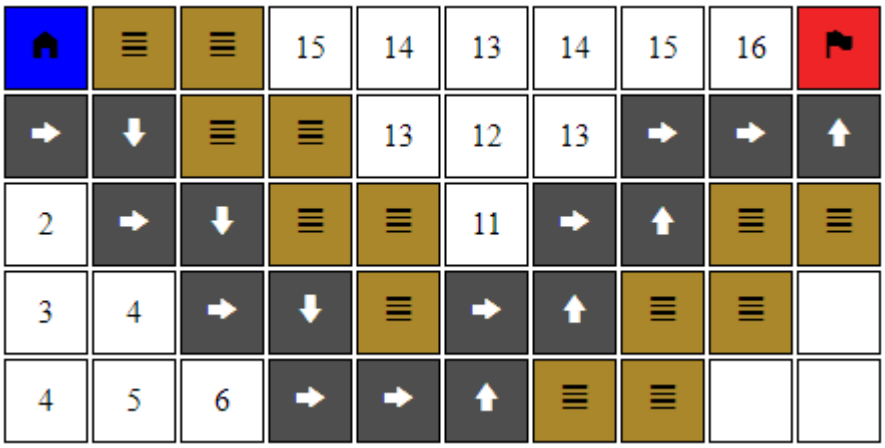
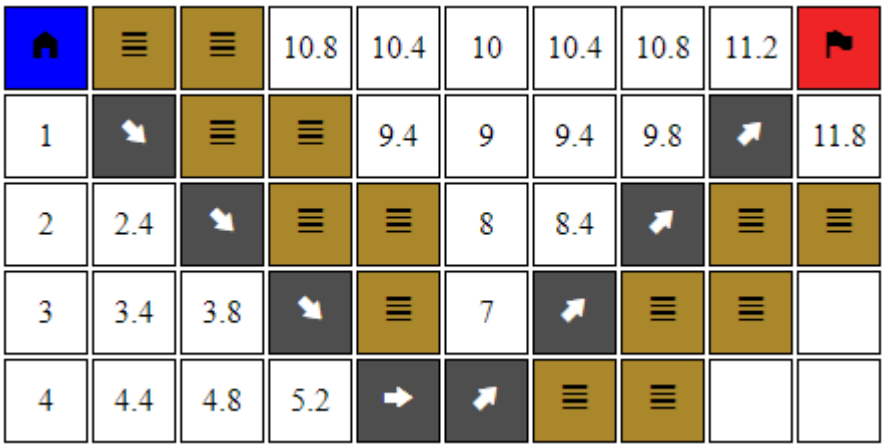


Рисунок 3.14 – Шлях із ортогональним режимом

Шлях було успішно знайдено. Після збереження та загрузки цієї карти знов включаємо трасування, але цього разу із діагональним режимом, і система повертає такий шлях – рисунок 3.15.



3.15 – Шлях із діагонально-ортогональним режимом

Шлях, котрий система повернула із діагональним режимом є набагато коротшим, і система по можливості обирає саме трасування по діагоналі, оскільки такий шлях є найбільш вдалим на цій мапі. Також хвиля отримує збільшення на 1.4 при поширенні по діагоналі.

Далі тестуванню підлягає можливість змінювати вплив елементів прискорення та сповільнення. Як було перевірено раніше – стандартна величина є 0.5, за для прикладу є мапа із увімкненим покроковим режимом та встановлено максимальний індекс впливу блоків. На рис. 3.16 можна побачити

різницю між стандартними настройками (а), та максимальними настройками (б).

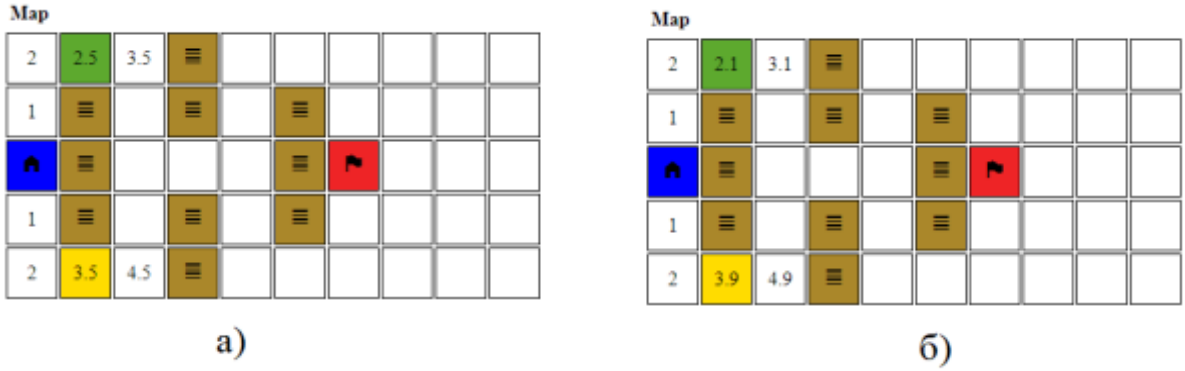


Рисунок 3.16 – Тестування зміни індексу прискорення та сповільнення

Отже, програмна система відповідає усім вимогам технічного завдання і може бути пристосована у реальних умовах

3.6 Висновок

В даному розділі дипломної роботи було розроблено структуру, інтерфейс, алгоритми. Як результат вдалось розробити програмну систему. У цьому розділі детально було розглянуто структуру системи. Ця структура має логічне поділення на 12 модулів. ґрунтуючись на цій структурі було створено і інші підрозділи. Після цього було розроблено інтерфейс із мінімалістичним дизайном, який добре пасує до складності програмної системи. У інтерфейсі повторюється та поліпшується логіка розділення системи на модулі, що поліпшує досвід користування. Розроблені алгоритми дозволили зосередити розробку навколо тонкощів обраної мови програмування. Як наслідок усіх цих підготовчих заходів розробка програмної системи саме на JavaScript стала швидкою та не потребувала багато років досвіду у розробці.

ВИСНОВОК

В результаті роботи над дипломною роботою були виконані наступні завдання.

1) Проаналізовані аналоги, у яких застосовується метод фронту хвилі: розглянуто їх можливості, виявлено недоліки та переваги;

2) Описані вимоги до системи. Виділено й узагальнено основні функції, які повинен виконувати система трасування шляху;

3) Розроблено основні алгоритми системи та інтерфейс;

4) Розроблено програмну систему, де для трасування шляху пристосовується метод фронту хвилі, та зворотне трасування, на мові програмування JavaScript;

5) Програмну систему було протестовано, і вона відповідає описаним вимогам та завданням;

У результаті ми маємо програмну систему, що дозволяє знаходити шлях у матриці елементів розмірами до 125 на 125, із можливістю зміни масштабу, що також дозволяє оперувати із матрицями менших розмірів. За допомогою вбудованого редактору є можливість визначити початок шляху, кінець шляху, перешкоди та місця більшого чи меншого пріоритету. Також мається можливість застосувати діагональний метод розповсюдження хвилі, увімкнути режим «крок-за-кроком», та зберегти чи загрузити отриману мапу. Інтерфейс системи є мінімалістичним, але успішно виконує свою задачу.

Система може бути вдосконалена у короткий проміжок часу, оскільки основні алгоритми винесені в окремі функції.

ПЕРЕЛІК ПОСИЛАНЬ

1. Опарин А. С. Анализ качества трассировки печатных плат. – 2012. – 124 с.
2. Уваров А. С. Автотрассировщики печатных плат. – М.: ДМК Пресс, 2016. – 288 с.
3. Пошук у ширину [Електронний ресурс]. URL: https://studopedia.com.ua/1_47182_rozvyazannya.html (дата звернення: 16.05.2020)
4. Лекція 27. Алгоритми пошуку найкоротших шляхів [Електронний ресурс]. URL: https://studfile.net/preview/3759996/ (дата звернення: 17.05.2020)
5. Программы для радиолюбителя [Електронний ресурс]. URL: https://cxem.net/programs.php (дата звернення: 13.05.2020)
6. [Електронний ресурс]. URL: http://qiao.github.io/PathFinding.js/visual/ (дата звернення: 15.05.2020)

ДОДАТОК А

Код розробленої програмної системи трасування маршрутів

```
<!DOCTYPE HTML>
<html>
<head>
   <title>Danylo Shevchuk Bachelor</title>
   <link rel="stylesheet" href="foundation-icons.css"/>
   <meta charset="utf-8">
   <style>
      table td {
         padding: 5px;
         border: 1px solid black;
         width: 30px;
         height: 30px;
         text-align: center;
         font-size: 100%;
      }
      input.bonus {
         width: 45px;
      }
      input.penalty {
         width: 45px;
      }
      p.h {
         font-weight: bold;
      }
      p {
         margin: 5px;
      }
      div {
         padding: 5px;
         border: 1px solid black;
         margin: 5px;
      }
      td#empty {
         background-color: #ffffff;
      }
```

```
        td#wall {
            background-color: #aa872a;
        }
        td#accelerator {
            background-color: #5da92e;
        }
        td#slower {
            background-color: #ffdc00;
        }
        td#start {
            background-color: #0000ff;
        }
        td#finish {
            background-color: #ee2426;
        }
        td#path {
            background-color: rgba(31, 31, 31, 0.79);
            color: #ffffff;
        }
        i {
            font-size: 18px;
        }
        select option#hidden {
            display: none;
        }
    </style>
    <!--[if lte IE 8]>
    <script
src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script><![endif]-->
</head>
<body>
<div id='tableNavigator'>
    <div id='tableContainer' style="border: 0; padding: 0; margin: 0">
        <p class='h'> Map </p>
    </div>
    <p class="h"> Change Map Scale </p>
    <input type='button' class='scale' id='plus' value='+'/> <input type='button'
class='scale' id='minus' value='-'/>
```

```
    <br>
    <p class="h"> Create, Export or Import Map </p>
    <input type='button' class='file' id='newMap' value='Create New Map'/> <input type='button' class='file'
                                                                                  id='saveMap'
                                                                                  value='Export Map to file...'
                                                                                  title="Wave indexes will not save, so you can re-use this map after re-uploading it"
                                                                                  disabled/>
    <p> Or just Import your pre-made map! </p>
    <input type='file' class='file' id='loadMapImport' value='Import Map from .json file...' accept=".json"/>
    <br>
    <input type='button' class='file' id='loadMap' value='Load Map...' style="margin-top: 4px"/>


</div>
<div>
<p id='modSelectorContainer'>
<p class='h'> Edit mode selector </p>
    <p><input type='radio' name='mod' value='empty' checked> Empty <i class="trash" style="color: #000000"></i></p>
    <p><input type='radio' name='mod' value='wall'> Wall <i class="align-justify" style="color: #aa872a"></i></p>
    <p><input type='radio' name='mod' value='accelerator'> Accelerator <i class="fast-forward" style="color: #5da92e"></i> cell bonus - <input type="number" step=".01" min=".1" max=".9" id="accelerator" value=".5" class="bonus"></p>
    <p><input type='radio' name='mod' value='slower'> Slower <i class="mountains" style="color: #ffdc00"></i> cell penalty + <input type="number" step=".01" min=".1" max=".9" id="slower" value=".5" class="penalty"></p>
    <p><input type='radio' name='mod' value='start'> Start <i class="home" style="color: #0000ff"></i></p>
    <p><input type='radio' name='mod' value='finish'> Finish <i class="flag" style="color: #ee2426"></i></p>
</div>
```

```
<div id='buttonsContainer'>
   <p class='h'> Route tracing </p>
   <input type="button" id="start" value="Start" disabled/>
   <p><input type='checkbox' name='step' value='finish'> Step-by-step mode</p>
   <p><input type='checkbox' name='direction' value='finish'> 8 directional mode</p>
   <input type="button" id="next" value="Next Step" disabled/>
</div>

<script>
   const mods = ['empty', 'wall', 'accelerator', 'slower', 'start', 'finish', 'path'];
   const modsContent = [false, 'align-justify', 'fast-forward', 'mountains', 'home', 'flag'];
   let config = {
      mapSize: {
         x: 7,
         y: 7,
         z: 4
      }
   };
   let startPosition = null;
   let finishPosition = null;
   let map = [];
   let modButtons = document.getElementsByName('mod');
   let currentMod = 'empty';
   let acceleratorBonus = 0.5;
   let slowerPenalty = 0.5;
   initModSelector();
   let stepMode = false;
   let diagonalMode = false;
   const startButton = document.getElementById('start');
   initStarter();
   let currentScale = 10;
   let currentCellStyle = {
      width: '30px',
      height: '30px',
      fontSize: '100%',
   };
```

```
initScaleMenu();
initSaveMenu();
let table;
function init(newMapNeeded) {
    startButton.disabled = false;
    if (newMapNeeded) {
        map = [];
        initMap();
    }
    updateTable(map);
}
function initModSelector() {
    for (let i = 0; i < modButtons.length; i++) {
        modButtons[i].onchange = selectChange;
    }
    function selectChange() {
        currentMod = this.value;
    }
    document.getElementById('accelerator').onchange = acceleratorOnChange;
    function acceleratorOnChange() {
        acceleratorBonus = this.value;
    }
    document.getElementById('slower').onchange = slowerOnChange;
    function slowerOnChange() {
        slowerPenalty = this.value;
    }
}
function initStarter() {
    startButton.onclick = () => {
        if (checkStartEndReady()) {
            stepSwitches[0].disabled = true;
            directionSwitches[0].disabled = true;
            document.getElementById('accelerator').disabled = true;
            document.getElementById('slower').disabled = true;
            if (stepMode) {
                nextButton.disabled = !stepMode;
                nextButton.click();
                for (let i = 0; i < modButtons.length; i++) {
```

```
                modButtons[i].disabled = true;
            }
        } else {
            waveStart()
        }
    }
};
const nextButton = document.getElementById('next');
nextButton.onclick = () => {
    if (checkStartEndReady()) {
        startButton.disabled = true;
        stepSwitches[0].disabled = true;
        directionSwitches[0].disabled = true;
        nextStep();
    }
};
const stepSwitches = document.getElementsByName('step');
stepSwitches[0].addEventListener('change', () => {
    stepMode = !stepMode;
    console.log('stepMode ===', stepMode)
});
const directionSwitches = document.getElementsByName('direction');
directionSwitches[0].addEventListener('change', () => {
    diagonalMode = !diagonalMode;
    console.log('diagonalMode ===', diagonalMode)
});
}
function initScaleMenu() {
const plus = document.getElementById('plus');
plus.onclick = () => {
    currentScale += 1;
    if (currentScale >= 20) {
        plus.disabled = true;
    }
    if (currentScale > 1) {
        minus.disabled = false;
    }
    applyScale();
```

```
    };
    const minus = document.getElementById('minus');
    minus.onclick = () => {
        currentScale -= 1;
        console.log('new scale', currentScale);
        if (currentScale < 20) {
            plus.disabled = false;
        }
        if (currentScale <= 1) {
            minus.disabled = true;
        }
        applyScale();
    };

}
function initSaveMenu() {
    const newMap = document.getElementById('newMap');
    newMap.onclick = () => {
        const x = askToEnterSize('Enter the heights of the map you want to create:');
        const y = askToEnterSize('Enter the width of the map you want to create:');
        config = {
            mapSize: {
                x: x,
                y: y,
                z: 4
            }
        };
        newMap.disabled = true;
        loadMap.disabled = true;
        saveMap.disabled = false;
        loadMapImport.disabled = true;
        init(true);
    };
    const loadMapImport = document.getElementById('loadMapImport');
    const loadMap = document.getElementById('loadMap');
    loadMap.onclick = function () {
        let files = loadMapImport.files;
        console.log(files);
```

```
    if (files.length <= 0) {
        alert('Chose files first!');
        return false;
    }
    let fr = new FileReader();
    fr.onload = function (e) {
        let result = JSON.parse(e.target.result);
        console.log(result);
        if (result.config && result.savedMap) {
            map = result.savedMap;
            config = result.config;
            finishPosition = result.finishPosition;
            startPosition = result.startPosition;
            newMap.disabled = true;
            loadMap.disabled = true;
            saveMap.disabled = false;
            loadMapImport.disabled = true;
            init(false);
            console.log(map);
        } else {
            alert('There is something wrong with your file!')
        }

    };

    fr.readAsText(files.item(0));
};
const saveMap = document.getElementById('saveMap');
saveMap.onclick = () => {
    let mapToSave = Array.from(map);
    for (let i = 0; i < mapToSave.length; i++) {
        const column = mapToSave[i];
        for (let j = 0; j < column.length; j++) {
            const cell = column[j];
            if (cell.data?.arrow) {
                column[j] = {
                    mode: cell.mode,
                    position: cell.position,
```

```
                    content: modsContent[cell.mode],
                    // waveIndex: null,
                }
            } else {
                column[j] = {
                    mode: cell.mode,
                    position: cell.position,
                    content: cell.content,
                    // waveIndex: null,
                }
            }
        }
    }
    let dataStr = "data:text/json;charset=utf-8," +
encodeURIComponent(JSON.stringify({
        savedMap: mapToSave,
        config: {
            mapSize: {
                x: config.mapSize.x,
                y: config.mapSize.y
            }
        },
        finishPosition: finishPosition,
        startPosition: startPosition
    }));
    let downloadAnchorNode = document.createElement('a');
    downloadAnchorNode.setAttribute("href", dataStr);
    downloadAnchorNode.setAttribute("download", "saved_map.json");
    document.body.appendChild(downloadAnchorNode); // required for firefox
    downloadAnchorNode.click();
    downloadAnchorNode.remove();
};
}
function askToEnterSize(msg) {
    let size = Number(prompt(msg, '7'));
    while (!size || size % 1 || size <= 0 || size > 126) {
        size = Number(prompt('Please try other number', '7'));
    }
```

```
    return size;
}
function applyScale() {
    currentCellStyle.width = 30 * currentScale / 10 + 'px';
    currentCellStyle.height = 30 * currentScale / 10 + 'px';
    currentCellStyle.fontSize = currentScale * 10 + '%';
    currentCellStyle.iconSize = 18 * currentScale / 10 + 'px';
    updateTable(map);
}
function initMap() {
    for (let i = 0; i < config.mapSize.x; i++) {
        const row = [];
        for (let j = 0; j < config.mapSize.y; j++) {
            row.push({
                mode: 0,
                position: {
                    x: i,
                    y: j
                }
            });
        }
        map.push(row);
    }
}
function updateTable(twoDimArray) {
    table = getNewTable();
    for (let i = 0; i < twoDimArray.length; i++) {
        const column = twoDimArray[i];
        const tr = document.createElement('tr');
        for (let j = 0; j < column.length; j++) {
            const cell = column[j];
            const td = document.createElement('td');
            td.style.width = currentCellStyle.width;
            td.style.height = currentCellStyle.height;
            td.style.fontSize = currentCellStyle.fontSize;
            td.id = cell.data?.arrow ? mods[6] : mods[cell.mode];
            if (cell.content) {
                if (cell.waveIndex) {
```

```
                    if (cell.data?.arrow) {
                        if (cell.mode === 2) {
                            updateCell(td, '', '#5da92e');
                            addIcon(td, cell);
                        } else if (cell.mode === 3) {
                            updateCell(td, '', '#ffdc00');
                            addIcon(td, cell);
                        } else {
                            updateCell(td, '');
                            addIcon(td, cell);
                        }
                    } else {
                        if (cell.mode === 2) {
                            updateCell(td, cell.waveIndex, cell.data?.arrow ? '#5da92e' :
null);
                        } else if (cell.mode === 3) {
                            updateCell(td, cell.waveIndex, cell.data?.arrow ? '#ffdc00' :
null);
                        } else {
                            updateCell(td, '');
                            addIcon(td, cell);
                        }
                    }
                } else {
                    updateCell(td, '');
                    addIcon(td, cell);
                }
            } else {
                updateCell(td, cell.waveIndex);
            }
            td.addEventListener('click', function tdClick() {
                setCell(this, cell);
            });
            tr.appendChild(td)
        }
        table.appendChild(tr)
    }
}
```

```
    function updateCell(td, waveIndex, color) {
        if (waveIndex > 0) td.innerHTML = waveIndex % 1 ? (+waveIndex).toFixed(1)
: waveIndex;
        if (color) td.style.color = color;
    }
    function setCell(td, cell) {
        if (waveStarted) return;
        if (currentMod === 'start' && startPosition) {
            alert('There should be only one start, please select other block')
        } else if (currentMod === 'finish' && finishPosition) {
            alert('There should be only one finish, please select other block')
        } else {
            if (currentMod === 'start') {
                startPosition = {x: cell.position.x, y: cell.position.y};
            } else if (currentMod === 'finish') {
                finishPosition = {x: cell.position.x, y: cell.position.y};
            }
            if (td.id === 'start') {
                startPosition = null;
            } else if (td.id === 'finish') {
                finishPosition = null;
            }
            td.id = currentMod;

            updateCell(td, cell.waveIndex);
            const index = mods.indexOf(currentMod);
            cell.mode = index;
            cell.content = modsContent[index] || null;
            const icon = td.querySelectorAll('i');
            if (icon[0]) {
                icon[0].remove();
            }
            addIcon(td, cell);
        }
    }
    function addIcon(td, cell) {
        const icon = document.createElement('i',);
        icon.className = cell.content;
```

```
        icon.style.fontSize = currentCellStyle.iconSize;
        td.appendChild(icon);
    }
    function clearTable() {
        table = document.getElementById('table');
        if (table) {
            table.remove();
        }
    }
    function getNewTable() {
        const parent = document.getElementById('tableContainer');
        clearTable(parent);
        let table = document.createElement('table');
        table.id = 'table';
        parent.appendChild(table);
        return table;
    }
    let elementsQueue = [];
    let nextElements = [];
    let biggestWaveIndex = 0;
    let finishFound = false;
    let wayHomeFound = false;
    let currentCell = null;
    // step mode variables
    let waveStarted = false;
    let pathFinderStarted = false;
    function waveNeighbor(x, y, nextWaveIndex) {
        if (map[x] && map[x][y]) {
            const mode = map[x][y].mode;
            if (mode === 0 && (!map[x][y].waveIndex || map[x][y].waveIndex >
+nextWaveIndex.toFixed(1))) {
                map [x][y].waveIndex = +nextWaveIndex.toFixed(1);
                nextElements.push({x: x, y: y})
            } else if (mode === 2 && (!map[x][y].waveIndex || map[x][y].waveIndex >
+nextWaveIndex.toFixed(1) - +acceleratorBonus)) {
                map [x][y].waveIndex = +nextWaveIndex.toFixed(1) -
+acceleratorBonus;
                nextElements.push({x: x, y: y})
```

```
        } else if (mode === 3 && (!map[x][y].waveIndex || map[x][y].waveIndex >
+nextWaveIndex.toFixed(1) + +slowerPenalty)) {
            map [x][y].waveIndex = +nextWaveIndex.toFixed(1) + +slowerPenalty;
            nextElements.push({x: x, y: y})
        } else if (mode === 5) {
            finishFound = true;
            biggestWaveIndex = map [x][y].waveIndex =
+nextWaveIndex.toFixed(1);
        }
    }
  }
  function waveStep(position) {
    const x = position.x;
    const y = position.y;
    map[x][y].waveIndex = map[x][y].waveIndex || 0;
    const currentWaveIndex = map[x][y].waveIndex;
    [
      [x + 1, y],
      [x - 1, y],
      [x, y + 1],
      [x, y - 1],
    ].forEach(el => {
      waveNeighbor(el[0], el[1], currentWaveIndex + 1);
      if (finishFound) return false;
    });
    if (diagonalMode) {
      [
        [x + 1, y - 1],
        [x - 1, y + 1],
        [x + 1, y + 1],
        [x - 1, y - 1],
      ].forEach(el => {
        waveNeighbor(el[0], el[1], currentWaveIndex + 1.4);
        if (finishFound) return false;
      });
    }
  }
  function checkStartEndReady() {
```

```
    if (!startPosition) {
        alert('Please mark starting position before running');
        return false;
    } else {
        if (!finishPosition) {
            alert('Please mark finish position before running');
            return false;
        } else return true;
    }
}
function waveStart() {
    if (checkStartEndReady()) {
        waveStep(startPosition);
        waveStarted = true;
        while (nextElements[0]) {
            elementsQueue = nextElements;
            nextElements = [];
            while (elementsQueue[0]) {
                waveStep(elementsQueue[0]);
                elementsQueue.splice(0, 1);
                if (finishFound) break;
            }
            updateTable(map);
            console.log('step ended');
            if (finishFound) {
                break
            }
        }
        console.log('wave ended');
        if (finishFound) {
            findWayHome();
        } else {
            alert('Sadly to say, but I do not know da wae.');
            alert('But you can save the map and re-upload it later!');
        }
    }
}
function comeBackStep(position) {
```

```
        const arrowIconConfig = [
            'arrow-up',
            'arrow-down',
            'arrow-left',
            'arrow-right',
            // diagonal mode
            'arrow-up-right',
            'arrow-down-left',
            'arrow-up-left',
            'arrow-down-right',
        ];
        let neighborIndex;
        let lowestNeighbor;
        const x = position.x;
        const y = position.y;
        const currentBackIndex = map[x][y].waveIndex > -1 ? map[x][y].waveIndex :
biggestWaveIndex;
        if (map[x + 1] && map[x + 1][y] && map[x + 1][y].waveIndex > -1 && map
[x + 1][y].waveIndex < currentBackIndex) {
            neighborIndex = 0;
            lowestNeighbor = map[x + 1][y];
        }
        if (map[x - 1] && map[x - 1][y] && map[x - 1][y].waveIndex > -1 && map [x
- 1][y].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex :
currentBackIndex)) {
            neighborIndex = 1;
            lowestNeighbor = map[x - 1][y];
        }
        if (map[x] && map[x][y + 1] && map[x][y + 1].waveIndex > -1 && map [x][y
+ 1].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex :
currentBackIndex)) {
            neighborIndex = 2;
            lowestNeighbor = map[x][y + 1];
        }
        if (map[x] && map[x][y - 1] && map[x][y - 1].waveIndex > -1 && map [x][y
- 1].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex :
currentBackIndex)) {
            neighborIndex = 3;
```

```
            lowestNeighbor = map[x][y - 1];
        }
        if (diagonalMode) {
            if (map[x + 1] && map[x + 1][y - 1] && map[x + 1][y - 1].waveIndex > -1
&& map [x + 1][y - 1].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex
: currentBackIndex)) {
                neighborIndex = 4;
                lowestNeighbor = map[x + 1][y - 1];
            }
            if (map[x - 1] && map[x - 1][y + 1] && map[x - 1][y + 1].waveIndex > -1
&& map [x - 1][y + 1].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex
: currentBackIndex)) {
                neighborIndex = 5;
                lowestNeighbor = map[x - 1][y + 1];
            }
            if (map[x + 1] && map[x + 1][y + 1] && map[x + 1][y + 1].waveIndex > -
1 && map [x + 1][y + 1].waveIndex < (lowestNeighbor ?
lowestNeighbor.waveIndex : currentBackIndex)) {
                neighborIndex = 6;
                lowestNeighbor = map[x + 1][y + 1];
            }
            if (map[x - 1] && map[x - 1][y - 1] && map[x - 1][y - 1].waveIndex > -1
&& map [x - 1][y - 1].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex
: currentBackIndex)) {
                neighborIndex = 7;
                lowestNeighbor = map[x - 1][y - 1];
            }
        }

        if (lowestNeighbor && lowestNeighbor.mode !== 4) {
            lowestNeighbor.content = arrowIconConfig[neighborIndex];
            lowestNeighbor.data = {arrow: true};
        }

        return lowestNeighbor;
    }

    function findWayHome() {
```

```
    currentCell = map[finishPosition.x][finishPosition.y];
    while (currentCell && currentCell.mode !== 4) {
        currentCell = comeBackStep(currentCell.position);
        console.log('step back found');
        updateTable(map);
    }
    wayHomeFound = true;
    startButton.disabled = true;
    console.log('way home found');
    console.log(map);
}

function nextStep() {
    if (wayHomeFound) {
        alert('refresh page to start over')
    } else {
        if (finishFound) {
            if (pathFinderStarted) {
                findWayHomeStep();
            } else {
                currentCell = map[finishPosition.x][finishPosition.y];
                pathFinderStarted = true;
                findWayHomeStep();
            }
        } else {
            if (waveStarted) {
                continueWave();
            } else {
                startWaveStep()
            }
        }
    }
}

function startWaveStep() {
    waveStep(startPosition);
    waveStarted = true;
    updateTable(map);
```

```
}

function continueWave() {
    if (nextElements[0]) {
        elementsQueue = nextElements;
        nextElements = [];

        while (elementsQueue[0]) {
            waveStep(elementsQueue[0]);
            elementsQueue.splice(0, 1);
            if (finishFound) break;
        }
        updateTable(map);
        console.log('step ended');
        if (finishFound) {
            nextElements = [];
        }
    } else {
        if (!finishFound) {
            alert('Sadly to say, but I do not know da wae.');
            alert('But you can save the map and re-upload it later!');
            wayHomeFound = true;
        }
    }
}

function findWayHomeStep() {
    if (currentCell && currentCell.mode !== 4) {
        currentCell = comeBackStep(currentCell.position);
        console.log('step back found');
        updateTable(map);
    } else {
        wayHomeFound = true;
        startButton.disabled = true;
        console.log('way home found');
    }
}
console.log(map);
```

```
</script>
</body>
</html>
```